Tema 4.- Análisis de requisitos

4.1.- Introducción

Según vimos en el tema 2 todas las normas identificaban uno o varios procesos relacionados con el Análisis de requisitos. Del mismo modo, en el tema 3, hemos presentado diversos modelos de ciclo de vida para el desarrollo de software y, en todos ellos, se puede observar la existencia de una fase denominada Análisis de requisitos.

Entre las tareas que hay que realizar en esta fase están el estudio de las características y la función del sistema, la definición de los requisitos del software y del sistema del que el software forma parte, así como la planificación inicial del proyecto y, posiblemente, algunas tareas relacionadas con el análisis de riesgos.

Todas estas tareas deben realizarse al comienzo del proyecto, pero el principal problema que se nos presenta es que, en estos momentos iniciales, es difícil tener una idea clara o, al menos, es difícil llegar a expresarla de cuáles son los requisitos del sistema y del software, y llegar a comprender en su totalidad la función que el software debe realizar. Por esto, algunos de los modelos de ciclo de vida estudiados proponen enfoques cíclicos de refinamiento de los requisitos o incluso de todo el proceso de desarrollo de software.

El análisis de requisitos es el primer paso técnico del proceso de ingeniería del software. Es aquí donde se refina la declaración general del ámbito del software en una *especificación* concreta que se convierte en la base de todas las actividades de ingeniería del software que siguen. Algunos modelos de ciclo de vida distinguen una fase de análisis de requisitos y otra de especificación del sistema. En cualquier caso, el análisis del sistema concluye con la especificación de los requisitos del mismo.

El análisis se centra en los ámbitos de información, funcional y de comportamiento del problema en cuestión. Para comprender mejor lo que se requiere se divide el problema en partes y se desarrollan representaciones o modelos que muestran la esencia de los requisitos (modelo esencial) y, posteriormente, los detalles de implementación (modelo de implementación). Como resultado del análisis se desarrolla la *especificación de requisitos*, un documento que describe el problema analizado y muestra la estructura de la solución propuesta.

La forma de especificar un sistema tiene una gran influencia en la calidad de la solución implementada finalmente. Tradicionalmente los ingenieros de software han venido trabajando con especificaciones incompletas, inconsistentes o erróneas, lo que invariablemente lleva a la confusión y a la frustración en todas las etapas del ciclo de vida. Como consecuencia de esto, la calidad, la corrección y la completitud del software disminuyen

Las técnicas de análisis que veremos conducen a una especificación en papel o basada en ordenador (si utilizamos herramientas de análisis o CASE), que contiene descripciones gráficas y textuales (normalmente en lenguaje natural o alguna forma de pseudocódigo) de los requisitos del software. Por otro lado, la construcción de prototipos lleva a una especificación ejecutable, es decir, el prototipo sirve como representación de los requisitos. Por último, los lenguajes de especificación formal

conducen a representaciones de los requisitos que pueden ser analizadas o verificadas formalmente. Sin embargo, sea cual sea el método elegido, existen una serie de características comunes:

1. Realizan una abstracción de las características del sistema, es decir, consisten en desarrollar un modelo del mismo.
2. Representan el sistema de forma jerárquica, basándose en mecanismos de partición del problema y estableciendo varios niveles de detalle.
3. Definen cuidadosamente las interfaces del sistema, tanto las interfaces externas, que relacionan el sistema con su entorno, como de las internas, las que se establecen entre los distintos módulos definidos.
4. Sirven de base para las etapas posteriores de diseño y de implementación.
5. Plantean varios niveles de especificación. Requisitos de usuario, requisitos de sistema, esenciales y de implementación.
6. No prestan demasiada atención a la representación de las restricciones o de criterios de validación (exceptuando los métodos formales).

Es por esta última deficiencia por la que surge la necesidad de los métodos de especificación formal, especialmente en aquellos sistemas en los que la corrección, completitud o fiabilidad del software juegan un papel fundamental. Ejemplos de este tipo de sistemas pueden ser los protocolos de comunicación, el software de control de una central nuclear o el de gestión del tráfico aéreo.

4.2.- Primeras cuestiones

El análisis de requisitos constituye el primer intento de comprender cuál va a ser la función y ámbito de información de un nuevo proyecto. Indudablemente, los esfuerzos realizados en esta fase producen beneficios en las fases posteriores. Aunque el desarrollo de esta fase plantea muchas cuestiones empezaremos por lo más básico:

4.2.1.- Los requisitos.

Obviamente la primera pregunta que cabe hacerse es: ¿Qué es un requisito?. Éste se puede definir como: Una condición o capacidad que necesita el usuario para resolver un problema o conseguir un objetivo determinado. Por extensión las condiciones que debe cumplir o poseer un sistema o uno de sus componentes para satisfacer un contrato, una norma o una especificación (Piattini 2003).

En Sommersville encontramos una clasificación para los requisitos. En ella se distingue entre requisitos funcionales, no funcionales y requisitos de dominio:

- **Requisitos funcionales** Son declaraciones de los servicios que proveerá el sistema, de la manera en que éste reaccionará a entradas particulares y de cómo se comportará en situaciones particulares. En algunos casos, los requisitos funcionales de los sistemas también declaran explícitamente lo que el sistema no debe hacer.

- **Requisitos no funcionales** Son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo. sobre el proceso de desarrollo, estándares. etcétera.
- **Requisitos del dominio** Son requisitos que provienen del dominio de aplicación del sistema y que reflejan las características de ese dominio. Éstos pueden ser funcionales o no funcionales.

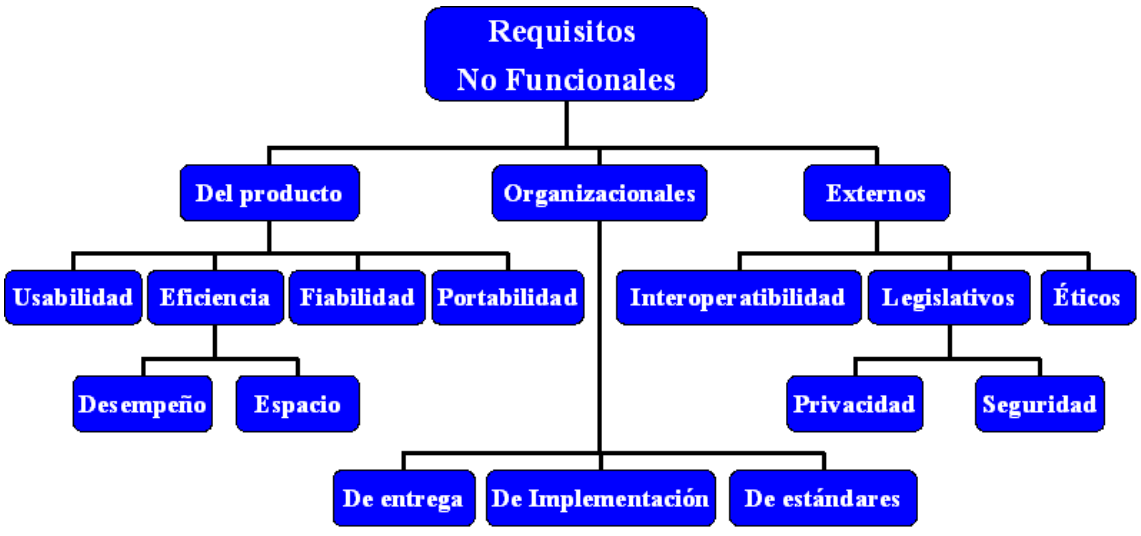
En realidad, la distinción entre estos tipos diferentes de requisitos no es tan clara como sugieren estas definiciones y con frecuencia al concretar requisitos de un tipo aparecen requisitos de otro.

Los requisitos funcionales de un sistema describen la funcionalidad o los servicios que se espera que éste provea. Estos, para un sistema de software, se expresan de diferentes formas. Cuando se expresan en lenguaje natural estos contienen con frecuencia ambigüedades e inconsistencias que son la fuente de muchos de los problemas de las ingenierías.

El sistema deberá proveer "visores adecuados" para que el usuario lea documentos en el almacén de documentos

Cuando un requisito funcional no se identifica o no se cumple el sistema no contiene toda su funcionalidad lo que supone una degradación del sistema que será tanto mayor cuantos más sean los requisitos incumplidos.

Los requisitos no funcionales son aquellos que no se refieren directamente a las funciones específicas que entrega el sistema, sino a las propiedades emergentes de éste como la fiabilidad, la respuesta en el tiempo y la capacidad de almacenamiento. Dicho de otro forma, definen las restricciones del sistema como la capacidad de los dispositivos de entrada/salida y la representación de los datos que se utiliza en las interfaces del sistema. Los requisitos no funcionales con frecuencia hacen referencia al sistema como un todo por lo que los fallos en ellos inutilizan el sistema lo que los convierte en más críticos que los requisitos funcionales.



La figura muestra una posible clasificación de los requisitos no funcionales en la que se aprecia claramente como estos no están únicamente relacionados con el producto

sino también con el entorno (organización y exterior) en el que se desarrolla y ejecuta la aplicación.

A menudo, los requisitos no funcionales entran en conflicto e interactúan con otros requisitos funcionales del sistema. En principio los requisitos funcionales y no funcionales se diferencian en el documento de requisitos. En la práctica, esto es difícil. Si se declaran de forma separada es difícil ver la relación entre ellos. Si se declaran en conjunto es difícil separar las consideraciones funcionales y no funcionales e identificar correctamente los requisitos. Se debe hallar un balance apropiado que permita resaltar claramente las propiedades emergentes del sistema, colocando, por ejemplo, estos requisitos en una sección aparte.

Finalmente los requisitos del dominio, son como su nombre indica, dependientes del dominio en el que se desarrolla la aplicación. Pueden ser requisitos funcionales nuevos, restringir los existentes o establecer cómo se deben ejecutar cálculos particulares. Los requisitos de dominio son importantes porque es imposible que el sistema trabaje satisfactoriamente si se incumplen. Los siguientes ejemplos aclaran el concepto.

> *Deberá existir una interfaz del usuario estándar para todas las bases de Datos, la cual tome como referencia el estándar Z39.50. (En un dominio de biblioteca.)*

> *La desaceleración del tren con alarma de luz roja se calculará como: Dmin=Dcontrol + Dgradiente; donde la Dgradiente es conocida y distinta según los tipos de trenes (Dominio de control ferroviario)*

La dificultad de trabajar con estos requisitos es que se expresan en un lenguaje específico de la aplicación que puede no ser bien comprendido por el analista. Además el usuario puede dejar fuera información necesaria porque para ellos es obvia.

4.2.2.- La duración del análisis.

El análisis de requisitos, debido a sus dificultades, es la fase que con más frecuencia tiende a expandirse indefinidamente en el tiempo, por tanto una de las primeras cuestiones que cabe realizar es: ¿Cuánto esfuerzo debemos dedicar al análisis?. Obviamente la respuesta dependerá de la naturaleza, el tamaño y la complejidad de la aplicación pero como guía se puede utilizar la regla de 40-20-40. En tal regla se propone que el análisis de requisitos y el diseño ocuparían el primer 40% mientras que las pruebas del sistema ocuparían el 40% final dejando el otro 20% para la codificación.

En la distribución del primer 40%, que es el que nos ocupa, la directriz que se plantea propone dedicar del 20% al 25% al diseño y entre el 15% y el 20% al análisis de requisitos. Finalmente podríamos afinar el tiempo de análisis dividiendo el tiempo entre un 10% y un 20% del esfuerzo al análisis del sistema y otro 10% o 20% al análisis de requisitos del software. Cabe destacar en este punto que no siempre será necesario realizar el análisis de sistema por lo que es posible una asignación total del 20%, máximo para el análisis, al análisis de requisitos del software

A pesar de estas consideraciones el esfuerzo que se le dedica normalmente al análisis es mucho menor. En la mayoría de los proyectos la regla no se cumple y la mayor parte del esfuerzo se va en la codificación, precisamente por la dificultad de realizar la codificación cuando no se ha hecho un buen análisis previo.

4.2.3.- El personal implicado.

En este punto lo primero que cabe plantearse es ¿quién tiene la responsabilidad de realizar el análisis?. La respuesta es fácil: el analista. Éste debe ser una persona con buena formación técnica y con experiencia. No obstante, el analista no trabaja de forma aislada sino en estrecho contacto con el personal de dirección, técnico y administrativo tanto del cliente como del que desarrolla el software.

En la actividad de análisis de requisitos el analista tiene un papel destacado en la comunicación entre Usuario/cliente y desarrolladores de software ya que, por un lado, los técnicos no conocen los detalles del trabajo de la empresa para la cual se va a desarrollar y, por otro, los usuarios no saben que información es necesaria para el desarrollo de una aplicación.

El analista es el principal puente que se establece entre el cliente y los desarrolladores. No sólo desarrollará su trabajo en la fase de análisis de requisitos sino que lo normal es que esté presente en todas las revisiones que se hagan a lo largo del desarrollo del proyecto. En su trabajo, el analista debe entrevistarse con múltiples personas, que tendrán una visión distinta del problema y de su solución incluso tendrán intereses contrapuestos.

Un buen analista, por tanto, debe facilidad de comunicación, debe ser capaz de extraer información relevante a partir de fuentes confusas o contradictorias, reorganizar esta información para sintetizar soluciones y debe servir de mediador entre las partes. Además de todo esto, debe tener los conocimientos técnicos y la experiencia necesarios para poder aplicar la informática a los problemas del cliente, debe conocer o ser capaz de asimilar conocimientos sobre el campo de actividad del cliente y debe tener claros los principios básicos de la ingeniería del software para poder incorporar estos principios a los requisitos del sistema de forma que se desarrolle software de calidad.

Pero la obtención y análisis de requisitos incluyen también a diferentes tipos de personas de la organización. El término **stakeholder** se utiliza para referirse a cualquier persona que tiene influencia directa o indirecta sobre los requisitos del sistema. Entre los stakeholders se encuentran los usuarios finales que interactúan con el sistema y todos aquellos en la organización que se verán afectados por dicho sistema. Los stakeholders también son los ingenieros que desarrollan o dan mantenimiento a otros sistemas relacionados, los administradores del negocio, los expertos en el dominio del sistema, los representantes de los trabajadores, etcétera.

4.2.4.- Dificultades del análisis de requisitos.

¿Por qué es una tarea tan difícil? Básicamente porque consiste en la traducción de unas ideas vagas de necesidades de software en un conjunto concreto de funciones y restricciones. Además el analista debe extraer información dialogando con muchas personas y cada una de ellas se expresará de una forma distinta, tendrá conocimientos

informáticos y técnicos distintos, y tendrá unas necesidades y una idea del proyecto muy particulares. Somersville lo concreta en las siguientes razones:

1. Los stakeholders a menudo no conocen realmente lo que desean obtener del sistema de cómputo excepto en términos muy generales; podrían encontrar difícil articular lo que quieren del sistema; podrían hacer demandas irreales debido a que no conocen el costo de sus peticiones.

2.-Los stakeholders de un sistema expresan los requisitos con sus propios términos de forma natural y con un conocimiento implícito de su propio trabajo. Los ingenieros de requisitos, sin experiencia en el dominio del cliente deben comprender estos requisitos.

3. Diferentes stakeholders tienen requisitos distintos y podrían expresarlos de varias formas. Los ingenieros de requisitos tienen que descubrir todas las fuentes potenciales de requisitos así como las partes comunes y en conflicto.

4. Los factores políticos influyen en los requisitos del sistema. Éstos provienen de los administradores quienes solicitan requisitos específicos para el sistema debido que esto les permitirá incrementar su influencia en la organización.

5. El entorno económico y de negocios en el que se lleva a cabo el análisis es dinámico. De forma inevitable cambia durante el proceso de análisis. Por lo que la importancia de ciertos requisitos puede cambiar. Emergen nuevos requisitos de nuevos stakeholders quienes no habían sido consultados previamente.

4.3.- Recogida inicial de requisitos

Según Raghavan[1] el proceso de análisis de requisitos se debería realizar siguiendo los siguientes 5 pasos.

- **Identificar las fuentes de información** (usuarios) relevantes para el proyecto.
- **Realizar las preguntas apropiadas** para comprender sus necesidades.
- **Analizar la información** recogida para detectar los aspectos que quedan poco claros.
- **Confirmar con los usuarios** lo que parece haberse comprendido de los requisitos.
- **Sintetizar los requisitos** en un documento de especificación apropiado.

Uno de los mayores problemas que surgen es cómo poner en contacto a usuarios y técnicos para establecer unos requisitos entendibles y aceptados por todos. Con este fin se plantean distintas técnicas de comunicación y recogida de información.

Las técnicas principales utilizadas para esta actividad son las siguientes [FLAATEN et al., 1989]:

[1] Piattini Pag 155-166

- **Entrevistas**. Es quizás la técnica más empleada y la que requiere una mayor preparación (y experiencia) por parte del analista. Es similar a una entrevista periodística en la que el desarrollador entrevista uno a uno a los futuros usuarios del software.
- **Desarrollo conjunto de aplicaciones (JAD).** Se crean equipos de usuarios y analistas que se reúnen para trabajar conjuntamente en la determinación de las características que debe tener el software para satisfacer las necesidades de los usuarios. Tiene una mayor probabilidad de éxito, ya que involucra al usuario en el proyecto, que lo aprecia como algo propio. A continuación desarrollaremos en detalle la Técnica para facilitar las especificaciones de una aplicación (TFEA)
- **Prototipado**. Consiste en la construcción de un modelo o «maqueta» del sistema que permite a los usuarios evaluar mejor sus necesidades, analizando si el prototipo que ven tiene las características de la aplicación que necesitan.
- **Observación**. Consiste en analizar in situ cómo funciona la unidad o el departamento que se quiere informatizar. Aporta grandes ventajas sobre las otras técnicas, ya que se pueden analizar mejor todos los detalles del proceso y se llega a captar el funcionamiento real de la empresa (que, a veces, no -coincide con las normas oficiales), así como el ambiente en el que se va a -desarrollar el proyecto y se va a instalar el futuro sistema.
- **Estudio de documentación**. En casi todas las organizaciones existen documentos que describen el funcionamiento del negocio, desde planes estratégicos hasta manuales de operación. El analista debe estudiar esta documentación para hacerse una idea de la normativa que rige la empresa. También es conveniente que recopile muestras de los impresos (que deben estar necesariamente rellenados para constatar que se usan realmente y para apreciar cómo se emplean) que se utilizan, ya que nos permiten conocer los datos que se manejan.
- **Cuestionarios**. Resultan útiles para recoger información de un gran número de personas en poco tiempo, especialmente en situaciones en las que se da una gran dispersión geográfica.
- **Tormenta de ideas (Brainstorming).** Algunos autores {RAGHAVAN et al., 1994] proponen la utilización de este tipo de reunión como medio para identificar un primer conjunto de requisitos en aquellos casos en los que no están muy claras todas las necesidades que hay que cubrir. Consiste en reuniones de cuatro a diez personas (usuarios) en las cuales, y en una primera fase, se sugieren toda clase de ideas sin juzgarse su validez, por muy disparatadas que parezcan. En una segunda fase, se realiza una análisis detallado de cada propuesta.
- **ETHICS8**. Constituye un método bastante evolucionado para fomentar la participación de los usuarios en los proyectos. Creado por E. Mumford en 1979 coordina la perspectiva social de los sistemas con su implementación técnica. Un sistema no tiene éxito si no se ajusta a los factores sociales y organizativos que rigen la empresa. Se busca la satisfacción de los empleados en el trabajo a través de estudios integrales (similares a los realizados por los especialistas en RR.HH. basados en factores como el conocimiento, la psicología, la eficiencia, la motivación, etc.). Los requisitos técnicos del sistema serán los necesarios para mejorar la situación de los empleados (y, por lo tanto, su productividad) en función de dichos análisis.

En la práctica es habitual utilizar combinaciones de diversas técnicas para recoger información de los usuarios. Los cuestionarios, la observación y el estudio de documentación son técnicas que, por sí solas, no suelen bastar para obtener la información necesaria para el análisis. Por eso, se suelen emplear en coordinación con las entrevistas y el TFEA, que analizaremos en los próximos apartados. El prototipado constituye una técnica bastante especial, ya que implica una forma distinta de concebir el desarrollo de software. No obstante, para crear un primer prototipo debe recurrir a las otras técnicas para tener un mínimo conocimiento de las necesidades del usuario.

4.3.1.- Método de trabajo TFEA

Mientras que los métodos clásicos se basan en entrevistas bilaterales (el analista y cada una de las partes) con lo que dejan para el analista toda la labor de organización y obtención de un consenso, últimamente se tiende a prácticas más relacionadas con el *brainstorming* en el que cada parte expone sus ideas y propuestas y se produce un debate de forma que las posiciones vayan acercándose sucesivamente hasta que se llegue a un consenso. Son las denominadas Técnicas para facilitar las especificaciones de una aplicación (TFEA)[2].

El método de trabajo sería el siguiente:

Inicialmente se llevan a cabo unas reuniones preliminares con el cliente (el que ha encargado el proyecto) con vistas a aclarar el ámbito del problema y los objetivos generales de una solución del mismo. Como resultado de estas reuniones, el analista y el cliente redactan una descripción breve del problema y los objetivos del proyecto y el esquema general de la solución.

A continuación se convoca una reunión en la que deben estar representados el analista, el cliente y el equipo de desarrollo. Los invitados a la reunión deben conocer previamente la descripción del problema y su solución redactada anteriormente y se les pide que elaboren una relación preliminar de los objetos o entidades que pueden identificar en el sistema o su entorno, las operaciones que se realizan o sirven para interrelacionar estos objetos, las restricciones que debe cumplir el sistema y los rendimientos que se esperan del mismo. Para conducir la reunión se nombra a un moderador neutral.

La reunión comienza discutiendo la necesidad y justificación del proyecto (es decir, debatiendo el documento preliminar). Una vez que todo el mundo esté de acuerdo en la necesidad de desarrollar el producto, se puede pasar a presentar cada una de las listas de objetos, operaciones, restricciones y rendimientos realizadas individualmente. A partir de estas listas individuales se crean listas combinadas, en las que se agrupa lo redundante pero no se elimina nada. En este punto está estrictamente prohibido el debate o las críticas.

Puede entonces comenzar la discusión sobre la lista combinada, en la que se pueden añadir nuevos elementos, eliminar otros o reescribirla de forma que refleje

[2] Pressman5, pag. 184

correctamente el sistema a desarrollar. Hay que realizar una *miniespecificación* de cada uno de los elementos de las listas, donde se defina brevemente cada uno de dichos elementos. El desarrollo de estas miniespecificaciones puede descubrir nuevos elementos o demostrar que alguno de ellos está incluido en otro. Se pueden dejar en el aire ciertos aspectos siempre y cuando se registre que deben ser tratados posteriormente.

Una vez concluida la reunión el analista debe elaborar un borrador de especificación del proyecto, combinando las miniespecificaciones de cada elemento, que servirá de base para todo su trabajo posterior.

Método de trabajo TFEA.

- *Redacción de un documento inicial:*
  - *Descripción del problema.*
  - *Objetivos.*
- *Propuesta de solución (si es posible).*
- *Creación de listas individuales de:*
  - *Objetos: del sistema y del entorno*
  - *Operaciones: relación entre los objetos*
  - *Restricciones*
  - *Rendimiento*
- *Creación de una lista conjunta*
- *Discusión de la lista conjunta*
- *Redacción de miniespecificaciones*
- *Redacción de un borrador de la especificación*

Este enfoque en equipo de la recogida inicial de requisitos proporciona numerosas ventajas, entre las que se pueden citar la comunicación multilateral de todos los involucrados en el proyecto, el refinamiento instantáneo y en equipo de los requisitos a partir de las ideas previas individuales y la obtención de un documento que sirva de base para el proceso de análisis.

Caso Práctico: Hogar Seguro

Descripción: Nuestras investigaciones indican que el mercado de sistemas de seguridad para el hogar está creciendo a un ritmo del 40% anual. Nos gustaría entrar en este mercado construyendo un sistema de seguridad para el hogar basado en un microprocesador que proteja y/o reconozca varias situaciones indeseables tales como irrupciones ilegales, fuego, inundaciones y otras. El producto, provisionalmente llamado *HogarSeguro*, utilizará los sensores adecuados para detectar cada situación, ha de ser programable por el usuario y llamará automáticamente a una agencia de vigilancia cuando se detecte alguna de estas situaciones.

EQUIPO TFEA:

Representantes de Marketing
Ingenieros de Software

Ingenieros de Hardware
Ingenieros Industriales (para la fabricación)
Coordinador

Listas de objetos:
Detectores de humos
Sensores en puertas y ventanas
Sensores de humo
Sensores de movimiento
Alarma acústica y visual
Panel de control
Pantalla
Teclado ...

Listas de acontecimientos (servicios):
Instalación de alarma
Vigilancia de sensores
Llamada de teléfono al centro de vigilancia
Programación del panel
Interacción con la pantalla de visualización
...

Restricciones:
Coste de fabricación < 200 €
Interfaz usable
Conexión a línea telefónica habitual
Conexión a línea telefónica inalámbrica
Funcionamiento mínimo independiente de electricidad
...

Rendimiento:
Respuesta a un acontecimiento < 1 seg.
Establecer listas de prioridades de actuación ante varios acontecimientos
...

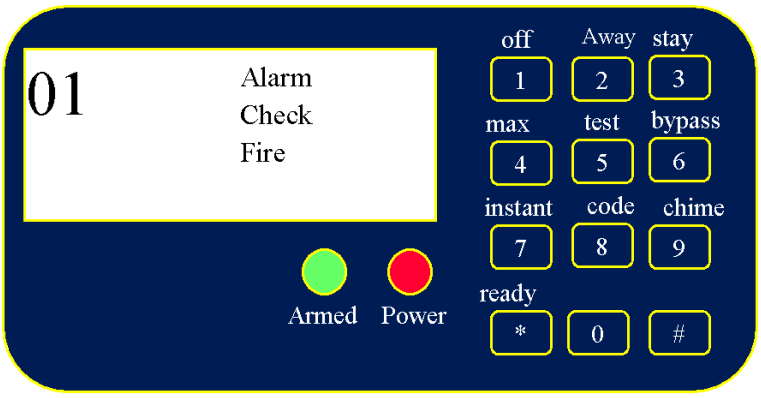
Pasos a seguir en la reunión reunión:

1.- Necesidad/Justificación
2.- Realización de listas conjuntas
3.- Miniespecificaciones para cada elemento de la lista
4.- Presentación de miniespecificaciones
5.- Descubrimiento de nuevos elementos
6.- Aspectos no abordables
7.- Primer borrador de especificaciones

Miniespecificación para el panel_de_control:

Montado en la pared

Tamaño aprox. 23x13 cm.
Teclado estándar de 12 teclas+teclas especiales
Pantalla LCD según gráfico adjunto
Interacción con usuario única por teclado
Software de ayuda para manejo
Conexión con todos los sensores



4.4.- Análisis de requisitos del sistema

El objetivo es conseguir representar un sistema en su totalidad, incluyendo hardware, software y personas, mostrando la relación entre los diversos componentes y sin entrar en la estructura interna de los mismos. En algunos casos se nos plantearán diferentes posibilidades y tendremos que realizar un estudio de cada una de ellas.

4.4.1.- Fases del análisis de sistemas.

El análisis del sistema comprende las siguientes fases o tareas:

- *Identificación de las necesidades del cliente.*
- *Asignación de funciones a cada uno de los elementos del sistema.*
- *Evaluación de la viabilidad del sistema.*
  - *Análisis económico y técnico del proyecto.*
- *Obtención de una definición del sistema que sea la base del trabajo posterior.*

4.4.1.1.- FASE: Identificación de las necesidades.

Para identificar las necesidades, el analista del sistema debe reunirse con el cliente o con su representante. Juntos, proceden a definir los objetivos del sistema, la información que se va a suministrar, la información que se va a obtener, las funciones y el rendimiento requerido.

El papel del analista de sistemas es el de definir los elementos de un sistema informático dentro del contexto del sistema en que va a ser usado y debe ser capaz de distinguir entre lo que *necesita* el cliente (lo que es para él imprescindible), y lo que *quiere* el cliente (aquello que le sería útil pero no imprescindible). El analista tiene que identificar las funciones del sistema y asignarlas a alguno de sus componentes. Para ello, parte de los objetivos y restricciones definidos por el usuario y realiza una representación de la función del sistema, de la información que maneja, de sus interfaces y del rendimiento y las restricciones del mismo.

En la mayoría de los casos, el proyecto empieza con un concepto más bien vago y ambiguo de cuál es la función deseada. Entonces el analista debe delimitar el sistema, indicando el ámbito de funcionamiento y el rendimiento deseados. Por ejemplo, en el caso de un sistema de control, no basta con decir algo así como *'el sistema debe reaccionar rápidamente en caso de que la señal de entrada supere los límites de seguridad'* sino que hay que definir cuáles son los límites de seguridad de la señal de entrada, en cuánto tiempo debe producirse la reacción y cómo ha de ser esta reacción.

Una vez que se ha logrado delimitar la función, el ámbito de información, las restricciones, el rendimiento y las interfaces del sistema, el analista debe proceder a la asignación de funciones. Las funciones del sistema deben de ser asignadas a alguno de sus componentes (ya sean éstos software, hardware o personas). El analista debe estudiar varias opciones de asignación (considerando, por ejemplo, la posibilidad de automatizar o no alguna de estas funciones), teniendo en cuenta las ventajas e inconvenientes de cada una de ellas (en cuanto a viabilidad, costes de desarrollo y funcionamiento y fiabilidad) y decidirse por una de ellas, o bien presentar un estudio razonado de las opciones a quienes tengan que tomar la decisión. Para explicar todo lo anterior podemos poner el siguiente ejemplo[3]:

Especificación informal del sistema de clasificación de paquetes.

El sistema de clasificación de paquetes debe realizarse de forma que los paquetes que se mueven a lo largo de una cinta transportadora sean identificados (para lo cual van provistos de un código numérico) y clasificados en alguno de los seis contenedores situados al final de la cinta. Los paquetes de cada tipo aparecen en la cinta siguiendo una distribución aleatoria y están espaciados de manera uniforme. La velocidad de la cinta debe ser tan alta como sea posible; como mínimo el sistema debe ser capaz de clasificar 10 paquetes por minuto. La carga de paquetes al principio de la cinta puede realizarse a una velocidad máxima de 20 paquetes por minuto. El sistema debe funcionar las 24 horas del día, siete días a la semana.

Función del sistema.

[3] (Pressman5, pags 81 y 176)

Realizar la clasificación de paquetes que llegan por una cinta transportadora en seis compartimentos distintos, dependiendo del tipo de cada paquete.

Información que se maneja.

Los paquetes disponen de un código numérico de identificación.

Debe existir una tabla o algoritmo que asigne a cada número de paquete el contenedor donde debe ser clasificado.

Interfaces del sistema.

El sistema de clasificación se relaciona con otros dos sistemas. Por un lado tenemos la cinta transportadora. Parece conveniente que el sistema de clasificación pueda parar el funcionamiento de la cinta o del sistema de carga de paquetes en la cinta, en caso de que no pueda realizar la clasificación (por ejemplo si se produce una avería). Por otro lado, el sistema deposita paquetes en los contenedores, pero no se establece ningún mecanismo de vaciado o sustitución de los contenedores si se llenan. El sistema debería ordenar la sustitución o vaciado del contenedor o esperar mientras un contenedor esté lleno.

Como la descripción realizada por el cliente no establece los mecanismos para solventar estas dos situaciones estos detalles deben ser discutidos con el cliente.

Rendimiento.

El sistema debe ser capaz de clasificar al menos 10 paquetes por minuto. No es necesario que el sistema sea capaz de clasificar más de 20 paquetes por minuto.

Restricciones.

El sistema debe tener un funcionamiento continuo. Por tanto, debemos evitar la parada del sistema incluso en el caso de que para alguno de los componentes del mismo se averíen.

El documento no indica restricciones sobre la eficacia del sistema, es decir, sobre cuál es el porcentaje máximo que se puede tolerar de paquetes que pueden ser clasificados de forma errónea. Estos detalles también deben ser aclarados con el cliente.

Asignación de funciones.

Podemos considerar tres asignaciones posibles:

Asignación 1.

*Esta asignación propone una **solución manual** para implementar el sistema. Los recursos que utiliza son básicamente personas, y se requiere además algo de documentación, definiendo las características del puesto de trabajo y del sistema de turnos y una tabla que sirva al operador para relacionar los códigos de identificación de los paquetes con el contenedor donde deben ser depositados.*

La inversión necesaria para poner en marcha este sistema es mínima, pero requiere una gran cantidad de mano de obra (varios turnos de trabajo y operadores de guardia) con lo que los costes de funcionamiento serán elevados. Además hay que tener en cuenta que lo rutinario del trabajo provocará una falta de motivación en los operarios, lo que a la larga se acabará traduciendo en un mayor absentismo laboral. Todos estos factores deben de tenerse en cuenta a la hora de elegir esta u otra opción.

Asignación 2.

*En este caso, la solución es automatizada. Los recursos que se utilizan son: hardware (el lector de códigos de barras, el controlador, el **mecanismo de distribución**), software (para el lector de códigos de barras y el controlador, y una base de datos que permita asignar a cada código su contenedor) y personas (si en caso de avería la distribución se va a hacer manualmente). Cualquiera de los elementos hardware y software tendrán la correspondiente documentación sobre cómo han sido construidos y un manual de usuario.*

Aquí si hay que realizar una cierta inversión, para comprar o desarrollar los componentes del sistema, pero los costes de funcionamiento serán sin duda menores (sólo el consumo de energía eléctrica). Hay que tener en cuenta que el uso de dispositivos mecánicos (el mecanismo de distribución) va a introducir unos costes de mantenimiento y paradas por avería o mantenimiento con una cierta frecuencia.

Asignación 3.

*Los recursos que utilizamos aquí son: hardware (el lector, el **brazo robot**), software (el del lector, el del robot, incluyendo la tabla o algoritmo de clasificación) y la documentación y manuales correspondientes a estos elementos.*

En este caso la inversión inicial es, sin duda, la más elevada. Los costes de funcionamiento son bajos pero hay que considerar también el coste de mantenimiento del robot, que posiblemente tenga que ser realizado por personal especializado. Los

únicos motivos que nos harían decidir por esta opción en vez de la anterior vendrían dados por una mayor velocidad, un menor número de errores o unas menores necesidades de mantenimiento o frecuencia de averías.

Por otra parte esta solución puede tener problemas de viabilidad (si no encontramos un brazo robot que sea capaz de atrapar los paquetes según pasan por la cinta).

Además de las tres opciones propuestas, el ingeniero de sistemas debe considerar también la adopción de *soluciones estándar* al problema. Hay que estudiar si existe ya un producto comercial que realice la función requerida para el sistema o si alguna de las partes del mismo pueden ser adquiridas a un tercero. Aparte de considerar el precio de estos productos habrá que tener también en cuenta los costes del mantenimiento y el riesgo que se asocia al depender de una tecnología que no es propia (¿es la empresa proveedora estable?, ¿cuál es la calidad de sus productos?) valorando todo esto frente a los riesgos asociados a realizar el desarrollo nosotros mismos.

La labor del ingeniero o analista de sistemas consiste, en definitiva, en asignar a cada elemento del sistema un ámbito de funcionamiento y de rendimiento. Después, el ingeniero del software se encargará de refinar este ámbito para el componente software del sistema y de producir un elemento funcional, que sea capaz de ser integrado con el resto de los elementos del sistema.

4.4.1.2.- FASE: Asignación de funciones.

Como ya habíamos visto, para cada elemento del sistema puede haber una o varias asignaciones posibles. Es necesario estudiar cada una de las alternativas desde el punto de vista económico y técnico y elegir cuál es la más adecuada.

Para elegir entre las distintas alternativas debemos fijar una serie de parámetros de evaluación. Normalmente los parámetros de evaluación serán de orden económico (coste de las inversiones, ahorro que se producirá con el sistema nuevo), pero también pueden tenerse en cuenta parámetros relacionados con la fiabilidad del sistema, su capacidad, su rendimiento o la mejora de calidad de los productos, si se trata de un sistema de producción. Hay que ordenar cada uno de estos parámetros de acuerdo a su importancia, lo que dependerá de los objetivos generales de cada proyecto, y ver cuál de las distintas alternativas obtiene una mejor evaluación de acuerdo con los parámetros establecidos. Cuando dos o más alternativas satisfagan los parámetros de evaluación principales, optaremos entre ellas observando los parámetros de segundo orden y así sucesivamente.

Por ejemplo, en el caso del sistema de clasificación de paquetes, la alternativa elegida dependerá de qué parámetros consideremos más importantes. Si queremos que la inversión inicial sea pequeña la mejor alternativa será la primera. Si pretendemos en cambio unos costes de funcionamiento moderados, valorando que el plazo de amortización sea corto, la mejor opción será la segunda. Si damos más importancia a la fiabilidad y al bajo mantenimiento del sistema, la mejor opción puede ser la tercera.

Sistema de clasificación de paquetes. Criterios de evaluación.

	Alternativa 1	Alternativa 2	Alternativa 3
Inversión inicial	Nula	Moderada	Grande
Coste funcionamiento	Grande	Moderado	Moderado
Tiempo amortización	Nulo	Moderado	Grande
Fiabilidad	Moderada	Pequeña	Grande
Mantenimiento	Nulo	Grande	Moderado

4.4.2.- Estudio de viabilidad.

Cualquier proyecto sería viable si dispusiésemos de recursos humanos, temporales y económicos ilimitados. Pero los recursos son siempre limitados: existen restricciones sobre el número de personas que se pueden dedicar (especialmente si se trata de personal del cliente), sobre cuánto dinero nos podemos gastar en el proyecto (si la inversión necesaria para el desarrollo es demasiado alta el sistema no compensará los ahorros que se produzcan con su uso) y sobre los tiempos de entrega (nadie compra software a cinco años vista, además, según pasa el tiempo aumentan las posibilidades de que cambien los requisitos).

Por esto, es conveniente estudiar la viabilidad del proyecto lo antes posible, puesto que así se pueden ahorrar meses de esfuerzos y miles de euros en el desarrollo de un proyecto que al final se muestre como inviable. El estudio de viabilidad está muy relacionado con el análisis de riesgos: si el riesgo de un proyecto es grande, se reducen las posibilidades de producir software de calidad, es decir, disminuye la viabilidad.

Viabilidad económica. Consiste en comparar los beneficios futuros de la utilización del sistema con los costes de su desarrollo. La justificación económica es normalmente la principal consideración a la hora de decidir realizar o no cualquier proyecto. Por esto, aparte de decidir la viabilidad o no viabilidad se necesita también un análisis económico completo (no sólo si va a producir beneficios sino qué beneficios va a producir, a qué plazo, etc.).

Viabilidad técnica. Consiste en determinar si es posible desarrollar o no el producto, teniendo en cuenta sus restricciones, basándonos en los recursos humanos y técnicos a nuestro alcance. Como los objetivos, funciones y rendimiento son confusos al inicio del proyecto, cualquier cosa puede parecer inicialmente viable. Debido a esto, es conveniente revisar el estudio de viabilidad técnica una vez que las especificaciones estén más claras. Además, el análisis de viabilidad técnica debe ser completado con un estudio técnico del sistema en proyecto, que permita determinar las características técnicas del nuevo sistema y qué mejoras introduce sobre el proceso actual.

Viabilidad legal. Consiste en determinar si el proyecto infringe alguna disposición legal sobre el derecho a la intimidad de las personas, las normas de seguridad en el trabajo, de calidad de los productos, las leyes de Copyright si se van a utilizar componentes comprados a terceros para integrarlos en el sistema o basarnos en ellos para el desarrollo del producto software, y otras.

Viabilidad de Plazos. Estudiada en el tema anterior se trata de calcular los plazos para la realización viable del proyecto o comprobar si los plazos a los que se ha comprometido quien firma el contrato permiten realizar una calendarización realista.

4.4.2.1.- Análisis económico.

El análisis económico del proyecto consistirá en señalar los costes de desarrollo del proyecto y compararlos con los beneficios que producirá una vez desarrollado.

Los costes de desarrollo pueden ser cuantificados fácilmente (inversiones en equipos, horas-hombre necesarias en las fases de análisis, diseño y codificación). Sin embargo, suele ser más difícil determinar los beneficios futuros que producirá el proyecto una vez listo para ser usado. Algunos de estos beneficios pueden ser tangibles (disminución del tiempo necesario para realizar determinadas tareas) pero otros muchos son intangibles (mayor satisfacción o cualificación profesional de los usuarios del sistema, incremento de la capacidad de gestión, etc.), por lo que puede ser difícil realizar comparaciones coste-beneficio directas.

Hay que tener en cuenta que la mayoría de los sistemas de procesamiento de la información se realizan teniendo como principal objetivo una mayor cantidad, calidad y rapidez de acceso a la información, de forma que se pueda realizar una mejor gestión de la empresa. Todos estos beneficios citados son intangibles y, aunque se traducirán sin lugar a dudas en beneficios tangibles (ahorro en los costes de producción o en las tareas administrativas, incremento de ventas como resultado de una mayor rapidez de respuesta a las necesidades del mercado, etc.) es muy difícil formular numéricamente esta relación. Por este motivo, los beneficios intangibles se incluirán en el análisis económico para reforzar la justificación del proyecto, pero esta justificación debe basarse en la medida de lo posible en los beneficios, medibles y demostrables que producirá el nuevo sistema.

Los costes y beneficios pueden ser directos o indirectos dependiendo de si es posible establecer una relación de los mismos con la implantación del software. Por ejemplo un aumento de la capacidad de gestión de un almacén es un beneficio directo de la implantación de un sistema informático para el control del mismo. El acondicionamiento del almacén para adaptarlo al nuevo sistema (redistribución de espacios, etc.) puede considerarse un coste indirecto.

La única forma de demostrar estos beneficios será comparando los modos de trabajo con el nuevo sistema con los modos de trabajo actuales. El nuevo sistema, cambiará los modos de trabajo de la empresa u organización en la que se instale de forma que se producirá:

- una disminución del tiempo necesario para realizar determinadas tareas.
- una menor necesidad de mano de obra para realizar el trabajo actual.
- un aumento de productividad, que permitirá aumentar la producción con la mano de obra actual.

Cualquiera de los tres puntos anteriores puede ser evaluado cuantitativamente, de forma que se puede determinar el ahorro que se producirá con la puesta en marcha del nuevo sistema.

Pongamos como ejemplo el sistema de clasificación de paquetes visto antes. Supongamos que el sistema actual es el que corresponde a la *Asignación 1* (es decir, es un sistema manual) y que el nuevo sistema es el indicado en la *Asignación 3* (utilización de un lápiz óptico y de un brazo robot).

Sistema de clasificación de paquetes. Análisis coste-beneficio

Costes del sistema actual: ***105.000 €/año***

Costes del nuevo sistema:

Inversión inicial: ***240.000 €***

Funcionamiento: ***12.000 €/año***

Mantenimiento: ***6.000 €/año***

Operatividad: ***95 %***

Total coste anual: ***23.256 €/año***

En primer lugar, calculemos los costes del sistema actual. Supongamos que para el funcionamiento en turno ininterrumpido, incluyendo operadores de guardia, necesitamos 7 operadores y que el coste salarial (incluyendo cargas sociales) de cada operador es de 15.000 €/año. Podemos calcular el coste/hora del sistema actual, que resulta ser de 12 €/hora, aproximadamente.

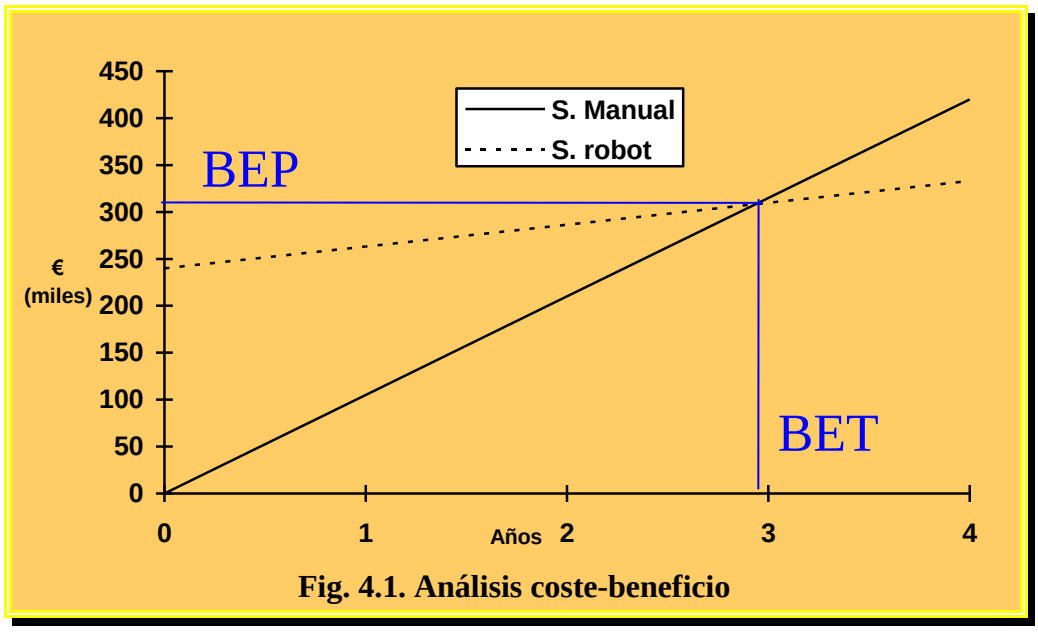


Fig. 4.1. Análisis coste-beneficio

Supongamos también que las inversiones necesarias para poner en marcha el sistema nuevo son de 240.000 €, que los costes de mantenimiento del brazo robot son de 6.000 €/año y los costes de funcionamiento (gastos de electricidad) son de 12.000 €/año. Además, debido a las paradas por mantenimiento y averías, el sistema nuevo solo estará útil el 95% del tiempo, debiendo efectuarse la clasificación manual el 5% del tiempo restante. Esto resulta en unos gastos de puesta en marcha de 240.000 € y unos gastos de funcionamiento de 23.256 €/año. Podemos representar los beneficios y costes acumulados del nuevo sistema en una gráfica (fig. 4.1) y calcular el tiempo de amortización (el punto donde se cortan ambas curvas: indica cuánto tiempo ha de pasar para que los beneficios del sistema nuevo compensen los costes iniciales de puesta en marcha). Este tiempo de amortización resulta ser de 2.9 años, aproximadamente.

4.4.2.2.- Análisis técnico.

Con el análisis técnico se pretende estudiar las características técnicas del nuevo sistema: capacidad, rendimiento, fiabilidad, seguridad, etc., de forma que se complemente el análisis de coste-beneficio con las mejoras técnicas que pueda proporcionar el sistema a los modos de trabajo de la empresa u organización donde se implante.

Para hacer un buen análisis técnico tendremos que modelar el sistema de alguna forma, y realizar un estudio analítico de las características del modelo propuesto o bien realizar algún tipo de simulación con el modelo.

Los resultados del análisis técnico son otro de los criterios que permitirán decidir si seguir o no con el proyecto. Si el riesgo técnico es alto, o si el modelo o las simulaciones muestran que el sistema en proyecto no va a conseguir substanciales mejoras sobre el sistema actual, podemos cancelar el proyecto.

4.4.3.- Representación de la arquitectura del sistema.

Como parte de los requisitos y diseño del sistema, éste tiene que modelarse como un conjunto de componentes y relaciones entre ellos que sirva de base para el trabajo posterior. Para ello se utiliza comúnmente una representación en diagrama de bloques que muestre los principales subsistemas y la interconexión entre ellos. Para estandarizar esta representación Pressman nos propone el uso de los diagramas de arquitectura a los que divide en cinco regiones[4].

[4] (Pressman5, pags 175 y 176)

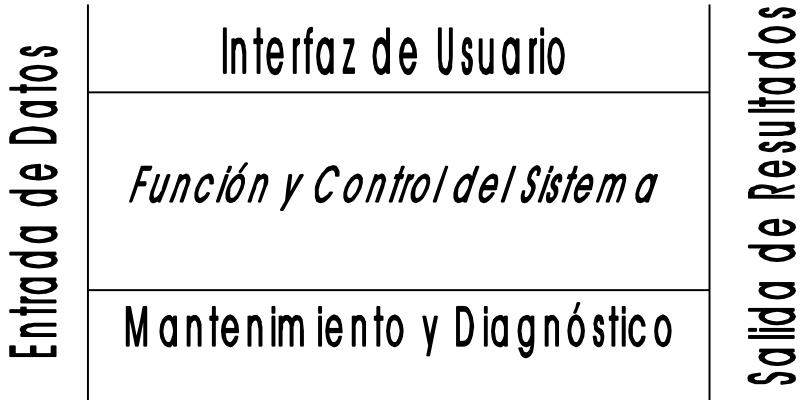


Figura 4.2.- Diagrama de arquitectura de un sistema

De esta forma los diagramas de arquitectura nos permitirán, además de representar las partes del sistema, identificar su entorno distinguiendo claramente sus interfaces externas. Si la complejidad del sistema así lo aconseja podemos utilizar estos diagramas formando una jerarquía de niveles, en el nivel superior representaremos el sistema mediante un diagrama de contexto, e iremos detallando más la arquitectura en sucesivos diagramas de flujo[5].

Diagrama de contexto.

El diagrama de contexto representa el sistema en relación con su entorno. Sirve para definir los límites del sistema y muestra todos los productores y consumidores de información del sistema.

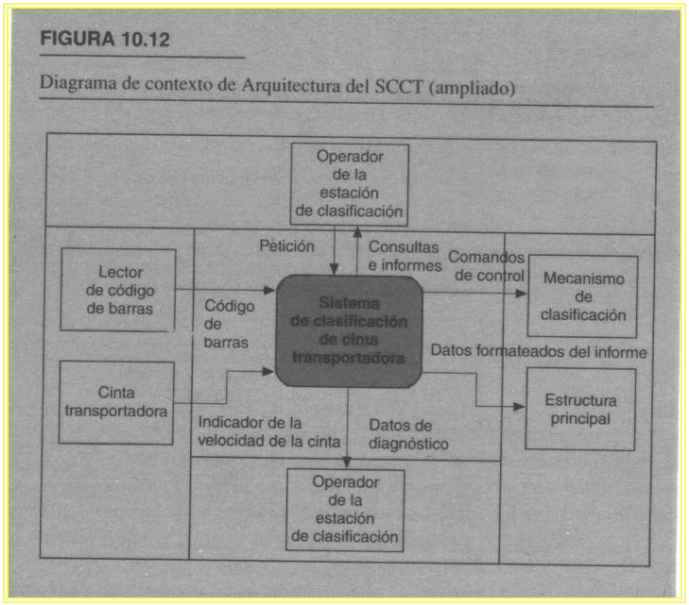


Figura 4.3.- Diagrama de contexto del sistema clasificador de paquetes

[5] Esta metodología se desarrollará en el tema 5.

El centro del diagrama de contexto estará ocupado por el sistema, representado por una caja de esquinas redondeadas. A su alrededor se situarán una serie de entidades o agentes externos (el entorno de sistema) representados mediante cajas de esquinas cuadradas. Cada agente externo representa un productor o consumidor de información del sistema. Cada agente externo se sitúa en la región del diagrama que le corresponda, según su papel sea el de productor, consumidor, usuario o supervisor.

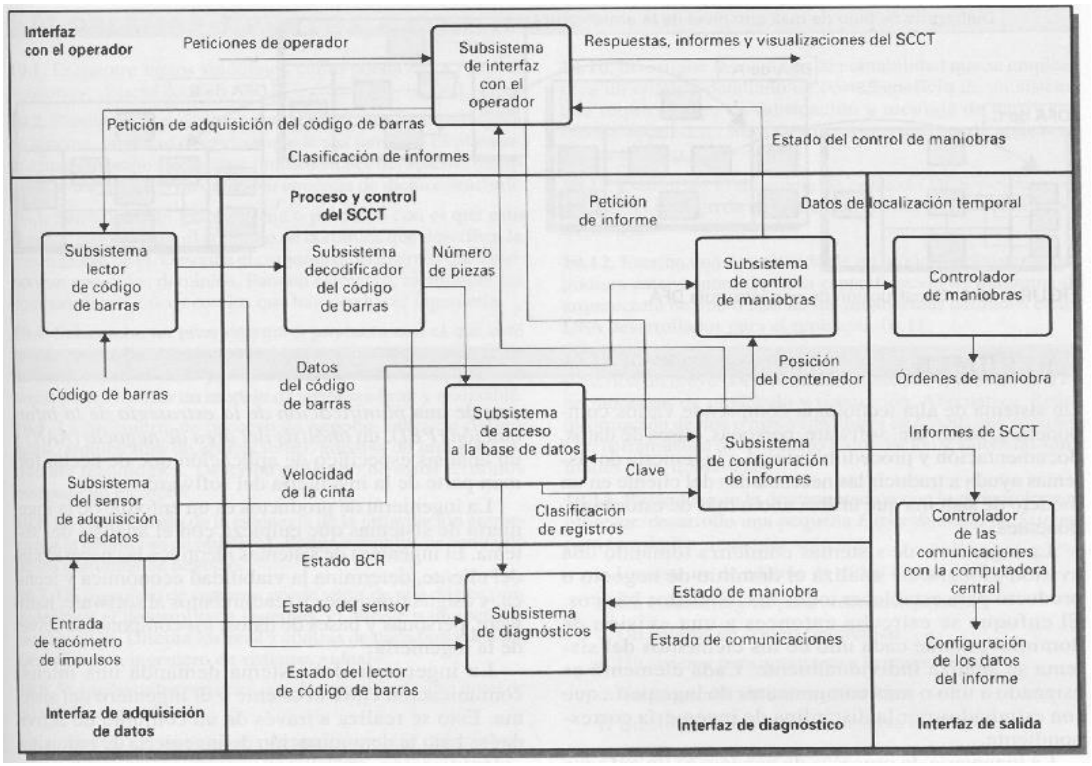
El sistema se relaciona con los agentes externos a través de flujos de información, representados mediante arcos orientados.

Diagramas de flujo.

Podemos describir la arquitectura del sistema en mayor grado de detalle a base de expandir el diagrama de contexto en una jerarquía de diagramas de flujo.

Cada subsistema del diagrama de contexto puede dar lugar a un diagrama de flujo, donde se describirá el sistema en mayor detalle, descomponiéndolo en nuevos subsistemas relacionados mediante flujos de información. Cada subsistema ocupa una región del diagrama dependiendo de cuál sea su función (adquisición de datos, salida de datos, interfaz, etc.).

Un detalle importante es que debe mantenerse la consistencia entre los diagramas de distinto nivel. Al expandir un determinado subsistema en un diagrama de flujo, los flujos de información que conectan dicho subsistema con otros o con agentes externos, deben figurar también (coincidiendo en número, sentido y nombre) en el diagrama de flujo resultado de dicha expansión. Estos flujos de información, tendrán un extremo libre, el que los conectaba con subsistemas que quedan fuera del ámbito del nuevo diagrama de flujo.



4.4.4.- Especificación del sistema.

La especificación del sistema es un documento que sirve de base para el análisis de cada uno de los componentes que intervienen en él, sean hardware, software o personas. Describe la función, rendimiento y restricciones que debe cumplir el sistema y limita cada uno de sus componentes, indicando el papel de cada uno de ellos y su interfaz.

La especificación del sistema es el documento que permite comprobar si el análisis realizado por el ingeniero de sistemas satisface las necesidades del cliente. Por este motivo, el cliente y el analista de sistemas deben revisar conjuntamente esta especificación para determinar si:

Evaluación inicial de la especificación

- *se ha delimitado correctamente el ámbito del proyecto.*
- *se ha definido correctamente la funcionalidad, las interfaces y el rendimiento.*
- *las necesidades del usuario y el análisis de riesgos justifican el desarrollo del proyecto.*
- *cliente y analista tienen la misma percepción de los objetivos del proyecto.*

Después de esta revisión con el usuario, es necesario realizar una evaluación técnica, para determinar si:

Evaluación técnica de la especificación

- *Las estimaciones de riesgos, coste y agenda se corresponden con la complejidad del proyecto.*
- *Todos los detalles técnicos (asignación de funciones, interfaces, rendimientos) están bien definidos.*
- *La especificación del sistema sirve de base para las fases siguientes (en concreto para la ingeniería de requisitos del software).*

Un posible formato de especificación del sistema podría ser el siguiente:

Especificación de requisitos del sistema.

I. Introducción.
- A. Ámbito y propósito del documento.
- B. Descripción general.
  1. Objetivos.
  2. Restricciones.

II. Descripción funcional y de datos.
- A. Diagrama de contexto de arquitectura.
- B. Descripción del DCA.

III. Descripción de los subsistemas.
- A. Especificación del diagrama de arquitectura para el subsistema *i*.
  1. Diagrama de flujo de arquitectura.
  2. Narrativa del módulo del sistema.
  3. Rendimiento.
  4. Restricciones de diseño.
  5. Asignación de componentes.
- B. Diccionario de la arquitectura.

IV. Resultados de la simulación del sistema.
- A. Modelo usado para la simulación.
- B. Resultados de la simulación.
- C. Aspectos especiales de rendimiento.

V. Aspectos del proyecto.
- A. Costes del proyecto.
- B. Agenda.
- C. Análisis de Viabilidad

VI. Apéndices.

4.5.- Análisis de requisitos del software

Como resultado de la fase de análisis de requisitos del sistema, se ha asignado una función y un rendimiento al componente software del mismo. Para conseguir esta función y rendimiento, el ingeniero del software debe construir - o adquirir - una serie de componentes software. Desgraciadamente, los componentes software están muy poco estandarizados, por lo que las dos únicas opciones serán el adquirir un sistema software comercial que cumpla con los requisitos - si este sistema existe y logramos encontrarlo - o desarrollar (o encargar el desarrollo) un sistema software a medida.

4.5.1.- Objetivos y actividades del análisis de requisitos del software.

El análisis de requisitos del software tiene como objeto desarrollar una representación del software que pueda ser revisada y aprobada por el cliente y es una tarea que sirve de enlace entre la asignación de funciones al software que se ha hecho en el análisis del sistema y el diseño del software.

Desde el punto de vista del analista de sistemas, el análisis de requisitos del software define con mayor precisión las funciones y rendimiento del software, las interfaces que ha de tener con otros componentes del sistema y las restricciones que debe cumplir.

Desde el punto de vista del diseñador, el análisis de requisitos proporciona una representación de la información, la función y el comportamiento del sistema que él se encargará de traducir en un diseño de datos y programas.

Por último, el análisis de requisitos, incluido en la especificación del proyecto, permite a todos (incluido aquí el cliente) valorar la calidad del software una vez que haya sido construido.

La labor del analista debe centrarse en el **qué**, no en el **cómo**. (¿qué datos debe manejar el sistema software?, ¿qué función debe realizar?, ¿qué interfaces debe tener?, y ¿qué restricciones tiene que cumplir?)

Pressman (Pressman 5ª 182-183) propone que el análisis de requisitos del software puede dividirse en cinco áreas de esfuerzo o actividades: (1) reconocimiento del problema, (2) evaluación y síntesis, (3) modelado, (4) especificación y (5) revisión.

Inicialmente, el, ingeniero del software o analista, estudia la especificación resultado del análisis del sistema, con el objetivo de comprender cuál es el papel del software en el contexto del sistema. El objetivo del analista es el reconocimiento de los elementos básicos del problema tal y como los percibe el cliente/usuario. Con este objetivo debe ponerse en contacto con el equipo técnico y de gestión del cliente, para poder conocer los objetivos básicos del software tal como los entiende el cliente. Con la información obtenida de la especificación del sistema y de las entrevistas con el cliente/usuario el analista debe ir definiendo y refinando los flujos y la estructura de la

información, la función del programa y su papel en el contexto del sistema, las características de las interfaces y las restricciones de diseño.

Con este conocimiento que se va adquiriendo del sistema software, el analista debe sintetizar una o varias soluciones al problema (modelos del problema), comprobando que se ajustan al plan del proyecto y a las necesidades del cliente. Este proceso de análisis del problema y síntesis de soluciones (modelos) debe proseguir hasta que el analista y el cliente acuerden una solución y se pueda especificar el software de forma adecuada para que se puedan efectuar las siguientes fases de desarrollo.

La síntesis de modelos permite entender mejor los flujos de información y la función de cada elemento del sistema software además estos modelos servirán de base para el diseño de software. A partir de estos modelos pueden desarrollarse prototipos del sistema software. Esto estará especialmente indicado cuando el cliente no esté seguro de lo que quiere realmente o cuando el analista necesite comprobar si una solución determinada resuelve efectivamente el problema. Los prototipos pueden ser utilizados por el cliente para refinar los requisitos o comprobar la validez de la solución propuesta.

La especificación debe indicar qué es lo que hay que hacer, pero no cómo hay que hacerlo. Debe ser una descripción de las necesidades y no una propuesta de solución. El cliente debe indicar qué características son imprescindibles y cuáles opcionales, y debe evitar describir la estructura interna del sistema, para no reducir la flexibilidad en la implementación. Características como, rendimiento, protocolos de comunicación, estándares de la IS (construcción modular, previsión de expansiones futuras, etc.), y, en algunos casos, la elección del soporte hardware y el lenguaje de implementación, pueden figurar en esta especificación. Otras decisiones de diseño, como el uso de un determinado algoritmo, no tienen nada que ver con el análisis y no son propiamente requisitos del sistema.

La especificación preliminar no es un documento inmutable. Normalmente será ambiguo, incompleto, incorrecto e inconsistente. Incluso aunque estén expresados de forma precisa, algunos de los requisitos reflejados pueden causar efectos secundarios no deseados (gran cantidad de información almacenada, tiempos de carga o de ejecución muy grandes) o llevar a costes de implementación demasiado grandes, frente a unos beneficios pequeños o innecesarios. Es necesario tomar la decisión de cumplir o no estos requisitos.

Otra manera de describir las actividades a realizar en la fase de análisis de requisitos está recogida por Piattini (pp 171) de Raghavan

1. **Extracción o determinación de requisitos**. El proceso mediante el cual los clientes o los futuros usuarios del software descubren, revelan, articulan y comprenden los requisitos que desean.
2. **Análisis de requisitos**. El proceso de razonamiento sobre los requisitos obtenidos en la etapa anterior, detectando y resolviendo posibles inconsistencias o conflictos, coordinando los requisitos relacionados entre sí, etc.

3. **Especificación de requisitos**. El proceso de redacción o registro de los requisitos. Para este proceso. puede recurrirse al lenguaje natural, lenguajes formales, modelos, gráficos, etc.
4. **Validación de los requisitos**. El proceso de confirmación, por parte de los usuarios o del cliente, de que los requisitos especificados son válidos, consistentes, completos, etc.

Igual que en la discusión anterior se precisa en este caso que estas actividades no tienen que realizarse en secuencia y que de hecho, habrá continuas iteraciones entre ellas. En cualquier caso la realización de estas actividades se apoya en distintas técnicas, algunas de ellas, como las de recogida de información ya han sido vistas pero la mayoría se discutirán en adelante.

4.5.2.- Principios del análisis de requisitos del software.

A lo largo de la historia de la Ingeniería del Software se han ido desarrollando diversos métodos de análisis del software, y diversas herramientas para facilitar el uso de estos métodos. Cada uno de los métodos tiene sus peculiaridades, utiliza una notación gráfica o textual distinta y tiene un campo de aplicación distinto. Sin embargo todos se basan en un conjunto de principios fundamentales.

Principios de análisis de requisitos del software

- *Identificar y representar el ámbito de información del problema.*
- *Modelar la información, la función y el comportamiento del sistema.*
- *Descomponer el problema de forma que se reduzca la complejidad.*
- *Avanzar desde lo más general a lo más detallado*

Los dos primeros puntos se refieren, por tanto, a conseguir representar el sistema y la información que maneja mediante un diagrama o una representación textual que permita comprender fácilmente qué información se maneja y qué funciones se realizan con esta información.

Los dos últimos se refieren a un método de trabajo *top-down*, que permita la división del problema en subproblemas, y vaya avanzando de los más general a lo más específico, de forma que la síntesis de la solución siga una estructura jerárquica.

El ámbito de información.

La tarea que realiza el software va a consistir siempre en procesar información: cualquier aplicación que consideremos responde a una estructura de entrada-procesamiento-salida, donde el software se encarga de procesar unos determinados datos de entrada para producir unos datos de salida. Estos datos pueden ser lógicos, numéricos, cadenas de caracteres, imágenes...

Pero en un sistema software, la información no está representada sólo por datos sino también por sucesos o eventos. En el sistema de clasificación de paquetes se procesa el número de identificación de cada paquete, obteniéndose el contenedor donde debe ser almacenado. Pero también se procesan sucesos: la clasificación de paquetes se

para si sucede que los contenedores se llenan; o se genera un informe de los paquetes procesados si el operador lo solicita. Los eventos son normalmente datos de tipo lógico (la activación o desactivación de la señal de un sensor, por ejemplo) y su procesamiento está ligado con el control del sistema.

En un sentido general, podemos decir que la información que maneja un sistema software se divide en datos y eventos. Los datos se refieren normalmente a la información que procesa el sistema y los eventos a cuándo debe procesarse esa información. El ámbito de información del sistema admite, por tanto, dos puntos de vista: por un lado, el flujo de datos (cómo se mueven los datos por el sistema y cómo se van transformando estos datos), y por otro, el flujo de control (cómo se controla el sistema y cuándo hay que realizar cada procesamiento).

En ambos casos se requiere el examen del dominio de la información y la creación de un modelo de datos que tiene tres aspectos: El contenido de la información, su estructura y el flujo de la información. El contenido hace referencia a los datos individuales que se manejan; la estructura, a como estos datos están organizados en entidades y como éstas están relacionadas; y el flujo de la información debe describir como cambian los datos y el control a medida que se mueven dentro del sistema lo que también incluye los flujos de control.

Modelado del sistema software.

Los modelos se utilizan en el campo de la ingeniería para entender mejor lo que se quiere construir. En la Ingeniería del Software se utilizan modelos para la información que transforma el software (modelos de datos), para las funciones que transforman esa información (modelos de procesos) y también para definir el comportamiento del sistema (modelos de control). Todos los métodos de análisis que veremos posteriormente son en realidad métodos de modelado de sistemas.

Los modelos que se realizan en la fase de análisis sirven para tres cosas:

Utilidad de los modelos

- *Ayudan al analista a entender la información, la función y el comportamiento del sistema, con lo que el análisis puede hacerse de forma más fácil y sistemática.*
- *Sirven de base para el trabajo del diseñador. La arquitectura de las aplicaciones y los datos debe corresponderse con los modelos del análisis.*
- *Sirven también para realizar la validación del producto una vez desarrollado. Por una parte, el sistema final debe comportarse como indica el modelo. Por otra, el sistema final permite comprobar la consistencia y la eficacia de la especificación.*

Para realizar un modelo podemos utilizar una notación gráfica (y el modelo estará representado mediante una serie de diagramas) o bien una notación textual (y tendremos modelos en lenguaje natural o en un lenguaje de especificación formal).

Descomposición del sistema.

La descomposición se utiliza para abordar problemas que son demasiado grandes o demasiado complejos para resolverlos directamente. Mediante

descomposición, el problema se divide en subproblemas más sencillos, que realizan una función más clara y que pueden entenderse más fácilmente. La descomposición se puede aplicar a los ámbitos de la información, de la función o del comportamiento.

Análisis de lo general a lo específico.

Descomponiendo un problema en subproblemas, y definiendo modelos de cada uno de estos problemas y subproblemas, llegaremos a establecer una jerarquía de modelos, que irán desde el más general (el de arriba del todo) a los más específicos (en los niveles bajos de la jerarquía). Cada modelo de la jerarquía produce, por descomposición de las funciones que realiza, una serie de modelos, cada uno de los cuales realiza una de estas funciones y que tendrán un mayor nivel de detalle.

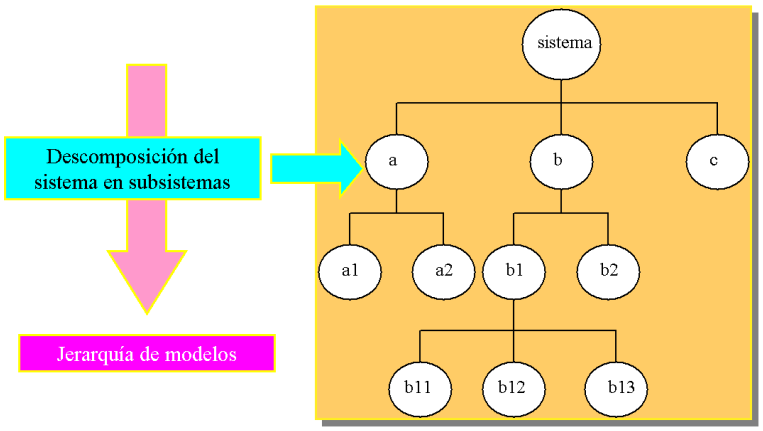


Figura 4.4.- Jerarquía de modelos

4.6.- Especificación de requisitos.

Para comprender qué es una Especificación, vamos a definir primero los siguientes términos [IEEE, 1990]:

- *Especificación es un documento que define, de forma completa, precisa y verificable, los requisitos, el diseño, el comportamiento u otras características de un sistema o componente de un sistema*.

La especificación del software es el documento que culmina las labores del análisis de requisitos. Debe contener una descripción detallada del ámbito de información, las funciones y el comportamiento asignados al software durante el análisis del sistema. Además debe contener información sobre los requisitos de rendimiento y restricciones de diseño y sobre las pruebas que se han de realizar para probar el software una vez que haya sido construido.

Por tanto, la Especificación se puede definir como la documentación de los requisitos esenciales (funciones, rendimiento, diseño, restricciones y atributos) del software y de sus interfaces externas [IEEE, 1990]. Los requisitos se representan de forma que conduzcan a una correcta implementación del sistema.

4.6.1.- Principios de la especificación

Pressman nos propone una serie de principios a seguir para la especificación:

Principios de la especificación.

- *La especificación debe modelar el dominio del problema.*
- *Es necesario separar funcionalidad e implementación.*
- *El lenguaje de especificación debe estar orientado al proceso.*
- *La especificación debe abarcar todo el sistema del que el software es parte.*
- *La especificación debe abarcar también el entorno del sistema.*
- *La especificación debe ser operativa.*
- *La especificación debe ser ampliable y tolerante a la incompletitud.*
- *La especificación debe estar localizada y débilmente acoplada.*

- **La especificación debe modelar el dominio del problema.**

La especificación del sistema ha de ser un modelo del dominio del problema, en vez de un modelo de diseño o implementación. Debe describir el sistema tal como es percibido por los expertos en el dominio de aplicación. Los elementos del sistema deben corresponderse con objetos reales de dicho dominio, ya sean individuos, máquinas u organizaciones, y las acciones que se realizan deben corresponderse con lo que realmente ocurre en el dominio del problema. Además, deben poder describirse en la especificación las reglas o leyes que gobiernan los objetos del dominio de aplicación. Algunas de estas leyes son restricciones sobre los estados del sistema, del tipo de 'dos objetos no pueden estar en el mismo lugar al mismo tiempo' y otras describen cómo responden los objetos cuando se actúa sobre ellos (por ejemplo las leyes del movimiento de Newton). En este caso, son parte inherente de las especificaciones del sistema.

Para que esto sea posible, el formato de la especificación y su contenido deben ser adecuados al problema.

- **Es necesario separar funcionalidad e implementación.**

Una especificación es, por definición, una descripción de lo que se quiere realizar, no de cómo se va a realizar o implementar. Como ejemplo de esto podemos tomar una especificación formal expresada mediante algún lenguaje declarativo: dado un conjunto de valores de entrada se produce otro de salida. Estas especificaciones se centran exclusivamente en el *qué* y no en el *cómo*. Esto es debido a que el resultado es una función matemática de la entrada. En estos casos de lo que se trata es de buscar alguna o todas las soluciones (modelos), tales que se cumpla P(entrada), donde P es un predicado que representa al sistema.

- **El lenguaje de especificación debe estar orientado al proceso.**

El ejemplo anterior muestra cómo se modelan sistemas que no están afectados por el entorno. Sin embargo, un caso más general debe considerar un sistema que interactúe con un entorno dinámico, modificando su comportamiento según los estímulos que recibe. Este podría ser el caso de un sistema empotrado. En este caso no podemos representar el sistema mediante una función matemática que relacione la entrada y la salida, sino que debemos emplear una descripción orientada al proceso, en la que la especificación del *qué* se consigue estableciendo un modelo del comportamiento deseado en términos de respuestas funcionales a distintos estímulos del entorno.

- **La especificación debe abarcar todo el sistema del que el software es parte.**

Un sistema está compuesto de partes que interactúan. El comportamiento de un componente específico, como es el software, sólo puede ser definido en el contexto del sistema completo. Por tanto, la especificación debe describir todos los componentes del sistema, estableciendo las interfaces entre estos componentes, y no sólo el componente software

- **La especificación debe abarcar también el entorno del sistema.**

Por el mismo motivo, hay que especificar también el entorno en el que opera el sistema. La especificación del entorno permite describir la interfaz del sistema de la misma forma que las interfaces de los componentes del mismo. De esta forma podemos representar sistemas dinámicos cuyo comportamiento varía dependiendo de los estímulos que reciban del entorno.

Esta especificación del entorno no se hace con vistas a implementarlo, puesto que ya nos viene dado y no podemos modificarlo, sino que sirve para definir los límites del sistema y su interfaz, permitiendo además probarlo.

- **La especificación debe ser operativa.**

La especificación ha de servir para determinar si una implementación concreta la satisface. Esto puede hacerse bien mediante la verificación de la corrección del sistema implementado, cosa que normalmente no puede demostrarse, o bien mediante una serie de casos de prueba elegidos arbitrariamente.

A partir de los resultados de una implementación sobre un conjunto arbitrario de datos de entrada, debe ser posible usar la implementación para validar estos resultados, a pesar de que la especificación no describa el *cómo* sino solamente el *qué*.

Este principio no establece que la especificación tenga que ser ejecutable, sino más bien que pueda ser utilizada para demostrar teoremas.

- **La especificación debe ser ampliable y tolerante a la incompletitud.**

Una especificación es un modelo o abstracción de un sistema real, por tanto nunca será completa, y puede desarrollarse a distintos niveles de detalle. Normalmente

el desarrollo de especificaciones se hace de forma incremental, estando en cada momento elementos del sistema parcialmente especificados. Aunque esto debilita el análisis, las herramientas de prueba de las especificaciones y de comprobación de que las implementaciones son correctas deben ser capaces de manejar especificaciones incompletas.

- **La especificación debe estar localizada y débilmente acoplada.**

Durante su desarrollo, una especificación sufre continuas modificaciones. Por este motivo, la estructura de la especificación debe permitir que estas modificaciones se realicen lo más fácilmente posible. El principio de localidad establece que si es necesario modificar un elemento de la especificación, esta modificación se realice sólo en un punto de la misma. El principio del acoplamiento débil establece que se puedan añadir y quitar partes de la especificación fácilmente.

Para cumplir estos dos principios la especificación ha de ser no redundante y modular, con interfaces breves y bien definidas.

4.6.2.- Características del documento de especificación.

Piattini en cambio nos propone una serie de características que la especificación debe cumplir para ser un documento útil.

1. No ambigua
2. Completa.
3. Fácil de verificar.
4. Consistente.
5. Fácil de modificar.
6. Facilidad para identificar el origen y las consecuencias de cada requisito.
7. Facilidad de utilización durante la fase de explotación y de mantenimiento.

1.- No ambigua

Un requisito ambiguo se presta a distintas interpretaciones. Por lo tanto, un documento de ERS no es ambiguo si y sólo si cada requisito descrito tiene una única interpretación. Esto implica que cada característica del producto final sea descrita utilizando un término único y, además, en los casos en los que un término usado en distintos contextos pueda tener distintos significados, debe incluirse en un ***glosario*** en el que se determina, de forma específica, su significado.

Con frecuencia, los requisitos se describen en lenguaje natural (por ejemplo, en castellano), lo que implica un gran riesgo, ya que este tipo de lenguaje cuenta con un alto potencial de ambigüedad. Los analistas que especifiquen los requisitos con un lenguaje natural deben poner especial atención en no caer en ambigüedades. Una alternativa que evita estos problemas es el uso de un lenguaje formal de especificación de requisitos

2.- Completa

Una ERS está completa si:

1. Incluye todos los requisitos significativos del software, ya sean relativos a la funcionalidad, ejecución, imperativos de diseño, atributos de calidad o a interfaces externas.
2. Define la respuesta del software a todas las posibles clases de datos de entrada y en todas las posibles situaciones. Nótese que es importante el especificar las respuestas tanto para entradas válidas como no válidas.
3. Está conforme con cualquier estándar de especificación que se deba cumplir. La adaptación a una norma puede implicar que si una sección particular del estándar no es aplicable al desarrollo del que se trata, la ERS debe incluir el correspondiente número de sección del estándar y una explicación que justifique su no aplicación.
4. Están etiquetadas y referenciadas en el texto todas las figuras, tablas y diagramas. También deben estar definidos todos los términos y unidades de medida.

Cualquier ERS que utilice la expresión «por determinar» (TBD: To Be Determined) no está completa. Sin embargo, hay veces que suele ser necesario utilizar un TBD y, en este caso, se debe acompañar de:

- Una descripción de las condiciones que han causado el TBD para que la situación pueda resolverse (por ejemplo, porque la administración aún no ha determinado el formato exacto de un impreso de pago de impuestos).
- Una descripción de qué hay que hacer para eliminar el TBD (por ejemplo, esperar a la publicación de un Real Decreto o un reglamento de impuestos).

3.- Fácil de verificar

Una ERS es fácil de verificar si y sólo si cualquier requisito al que se haga referencia se puede verificar fácilmente, es decir, si existe algún procedimiento finito y efectivo en coste para que una persona o una máquina compruebe que el software satisface dicho requisito. Un ejemplo de requisito difícil de verificar sería el siguiente:

«El programa no debe entrar nunca en un bucle infinito». La comprobación de este requisito es, incluso teóricamente, imposible. Si no se encuentra un método para determinar si el producto software satisface un requisito concreto, entonces se debe eliminar dicho requisito de la ERS. También puede ocurrir que un requisito no se pueda expresar de forma que se pueda verificar fácilmente cuando se redacta la ERS. Esto no constituye ningún problema si, posteriormente, se puede rescribir en el proyecto en una forma verificable.

4.- Consistente

Una ERS es consistente si y sólo si ningún conjunto de requisitos descritos en ella son contradictorios o entran en conflicto. Se pueden presentar tres tipos distintos de conflicto:

1. Dos o más requisitos pueden describir el mismo objeto real pero utilizan distintos términos para designarlo.
2. Las características especificadas de objetos reales pueden estar en conflicto. Por ejemplo, un requisito establece que todas las luces han de ser azules y otro que han de ser verdes.
3. Puede haber un conflicto lógico o temporal entre dos acciones determinadas. Por ejemplo, un requisito puede establecer que se deben sumar dos entradas y otro que han de multiplicarse.

5.- Fácil de modificar

Una ERS es fácilmente modificable si su estructura y estilo permiten que cualquier cambio necesario en los requisitos se pueda realizar fácil, completa y consistentemente. Esto implica que la ERS debe:

- Tener una organización coherente y manejable, con una tabla de contenidos, un índice y referencias cruzadas.
- No ser redundante; o sea, que el mismo requisito no aparezca en más de un lugar de la ERS.

La redundancia en sí no es un error, pero puede fácilmente conducir a errores. Ocasionalmente, la redundancia puede ayudar a la legibilidad de la ERS, pero seguramente provocará problemas cuando haya que actualizar el documento. Por ejemplo, supóngase que un requisito se define en dos lugares distintos de la ERS y posteriormente se decide que dicho requisito debe modificarse. Si el cambio sólo se realiza en uno de los lugares, la ERS quedará inconsistente.

Como la redundancia es difícil de evitar, lo mejor es crear referencias cruzadas entre los requisitos y los términos empleados para definirlos, facilitando así las posibles modificaciones en la ERS.

6.- Facilidad para identificar el origen y las consecuencias de cada requisito.

Se dice que una ERS facilita las referencias con otros productos del ciclo de vida si establece un origen claro para cada uno de los requisitos y si posibilita la referencia de estos requisitos en desarrollos futuros o en incrementos de la documentación. Se recomiendan dos tipos de referencias:

1. Referencias hacia atrás (esto es, a documentos previos al desarrollo). Depende de que los requisitos referencien explícitamente sus fuentes en documentos previos.
2. Referencias hacia adelante (esto es, a los documentos originados a partir de la ERS). Depende de que cada requisito de la ERS tenga un nombre o número de referencia único que sirva para identificarlo en futuros documentos.

Cuando un requisito de la ERS representa un desglose o una derivación de otro requisito, se debe facilitar tanto las referencias hacia atrás como hacia adelante en el ciclo de vida. Las referencias hacia adelante de la ERS son especialmente importantes para el mantenimiento de software. Cuando el código y los documentos son modificados, es esencial poder comprobar el conjunto total de requisitos que pueden verse afectados por estas modificaciones.

7.- Facilidad de utilización durante la fase de explotación y mantenimiento

La ERS también debe tener en cuenta las necesidades de mantenimiento, incluyendo la sustitución eventual del software, especialmente debido a que:

1.- El personal que no ha estado relacionado con el desarrollo del producto software se encarga del mantenimiento. Debe recordarse que las pequeñas correcciones suelen afectar sólo al código o al diseño detallado, por lo que se pueden documentar comentando adecuadamente el código y/o el diseño. Sin embargo, cuando las modificaciones son más profundas, es esencial actualizar la documentación del diseño y de los requisitos. Por lo tanto, para este último caso:
   - La ERS debe ser fácilmente modificable, como vimos con anterioridad.
   - La ERS debería prever un registro de las características especiales de cada componente, tales como:
     – Su criticidad (por ejemplo, en los componentes en los que un fallo puede causar mayor daño).
     – Su relación con necesidades temporales (por ejemplo, un listado o una pantalla que puede que no sean necesarios en poco tiempo).
     – Su origen (por ejemplo, la función X o la pantalla Y proceden íntegramente de un producto software existente).

2.- Gran parte de los conocimientos y de la información necesaria para el mantenimiento se dan por supuestos en la organización del desarrollo, pero suelen estar ausentes en la organización del mantenimiento. Si no se entiende la razón del origen de una función, es prácticamente imposible desarrollar el mantenimiento.

4.6.3.- Estructura para la ERS

Existen numerosos esquemas para la especificación del software, que varían dependiendo del método de análisis elegido. A titulo general, podemos proponer el siguiente:

Especificación de requisitos del software.

I. Introducción.
- A. Referencia del sistema.
- B. Descripción general.
  1. Objetivos.
  2. Restricciones.

II. Descripción de la información.
- A. Representación del flujo de la información.
  1. Diagramas de flujo de datos.
  2. Diagramas de flujo de control.
- B. Representación del contenido de la información.

III. Descripción funcional.
- A. Narrativa de procesamiento.
- B. Restricciones.

C. Requisitos de rendimiento.
D. Restricciones de diseño.

IV. Descripción del comportamiento.

A. Especificaciones de control.
B. Estados del sistema.
C. Eventos y acciones.
D. Restricciones de diseño.

V. Criterios de validación.

A. Descripción de las pruebas.
B. Respuesta esperada del software.
C. Consideraciones especiales.

VI. Apéndices.

Este esquema se corresponde con un análisis realizado siguiendo métodos estructurados, pero podríamos adaptarlo fácilmente al AOO.

La introducción describe el software en el contexto del sistema del que va a formar parte, indicando cuáles son los objetivos y restricciones del sistema software.

La sección de descripción de la información da una descripción detallada del problema que tiene que resolver el software, indicando los flujos de información que se mueven entre las partes del mismo y el contenido de estos flujos de información (mediante un diccionario de datos). Si usamos análisis estructurado, aquí irían los DFDs y DFCs. Si usamos AOO, incluiríamos aquí los diagramas de objetos.

En la sección de descripción funcional figurarían las definiciones de cada una de las funciones que componen el sistema, mediante la especificación de las primitivas de proceso. Además, hay que definir y justificar todas las restricciones, incluidas las de rendimiento y de diseño que deban satisfacer estas funciones. En el caso del AOO, figurarían aquí también los DFDs.

La sección de descripción del comportamiento debe incluir las especificaciones de control del sistema, normalmente en forma de Diagramas de Estados, definiendo también, si es necesario, las acciones y actividades a realizar y los eventos o sucesos que disparan las transiciones.

Por último, la sección de criterios de validación debe definir las pruebas que permitan determinar si una implementación del sistema satisface la función, restricciones y rendimiento establecidos en la especificación. Esto no es una tarea trivial, pues precisa un buen conocimiento de los requisitos del software y no debemos dejarla para cuando el software ya esté acabado, puesto que entonces se tiende a pasarla por alto o a realizar las pruebas precisamente en función del software construido y no en función del proyectado.

Al igual que la especificación del sistema, la especificación de requisitos del software debe sufrir una revisión realizada conjuntamente por el cliente y el ingeniero del software. Esta revisión puede hacerse a dos niveles. En primer lugar, para determinar si el sistema cumple con la función, restricciones y rendimiento requeridos

por el cliente, y para ver si ha de cambiarse algunas de las estimaciones de coste, riesgo o agenda definidas en el análisis del sistema.

El segundo nivel de revisión es mucho más detallado, y tiene por objeto detectar imprecisiones o ambigüedades en la especificación. Hay que revisar los términos vagos, como 'a veces', 'a menudo' o 'alguno' y las listas no exhaustivas, que acaban en 'etc.' o 'y otros' e intentar precisarlos. También hay que tener cuidado con las expresiones de certeza, como 'siempre', 'todos' o 'nunca' y comprobar si efectivamente son correctos (ej. el detalle de una factura siempre cabe en una página). En muchos casos, expresiones como 'obviamente' o 'por tanto' no son tan obvias e intentan ocultar una laguna en la especificación (ej. hoy es miércoles, por tanto, llueve).

4.7.- Validación de requisitos

Esta validación muestra que éstos son los que definen el sistema que el cliente desea. Tiene mucho en común con el análisis, ya que implica encontrar problemas con los requisitos. Sin embargo, son procesos distintos puesto que la validación comprende un bosquejo completo del documento de requisitos mientras que el análisis implica trabajar con requisitos incompletos. Las consecuencias y los costos de los errores en el documento de requisitos ya han sido discutidas sobre los ciclos de vida por lo que no se hará más hincapié en este punto.

Durante el proceso de validación de requisitos, se deben llevar a cabo diferentes tipos de verificación de requisitos en el documento de requisitos. Estas verificaciones incluyen:

1. *Verificaciones de validez:* Un usuario puede pensar que se necesita un sistema para llevar a cabo ciertas funciones. Sin embargo, el razonamiento y el análisis identifican que se requieren funciones adicionales y diferentes. Los sistemas tienen diversos usuarios con diferentes necesidades y cualquier conjunto de requisitos es inevitablemente un compromiso en el entorno del usuario.
2. *Verificaciones de consistencia:* Los requisitos en el documento no deben contradecirse. Esto es, no debe haber restricciones contradictorias o descripciones diferentes de la misma función del sistema.
3. *Verificaciones de integridad:* El documento de requisitos debe incluir requisitos que definan todas las funciones y restricciones propuestas por el usuario del sistema.
4. *Verificaciones de realismo:* Utilizando el conocimiento de la tecnología existente. los requisitos deben verificarse para asegurar que se pueden implementar. Estas verificaciones también deben tomar en cuenta el presupuesto y calendarización del desarrollo del sistema.
5. *Verificabilidad:* Para reducir las discusiones entre el cliente y el contratista, los requisitos del sistema siempre deben redactarse de tal forma que sean verificables. Esto significa que puede diseñarse un conjunto de verificaciones para demostrar que el sistema a entregar cumple esos requisitos.

Existen varias técnicas de validación de requisitos que pueden utilizarse en conjunto o de forma individual:

1. *Revisiones de requisitos* Los requisitos son analizados sistemáticamente por un equipo de revisores. Este proceso se discute en 1a siguiente sección.
2. *Construcción de prototipos* En este enfoque de validación, se muestra un modelo ejecutable del sistema a los usuarios finales y a los clientes. Éstos pueden hacer experimentos con este modelo para ver si cumple sus necesidades reales.
3. *Generación de casos de prueba* De forma ideal. los requisitos deben poder probarse. Si las pruebas para éstos se consideran como parte del proceso de validación, esto a menudo revela los problemas en los requisitos. Si una prueba es difícil o imposible de diseñar, por lo regular esto significa que los requisitos serán difíciles de implementar y deberían ser considerados nuevamente.
4. *Análisis de consistencia automático* Si los requisitos se expresan como un modelo del sistema en una notación estructurada o formal, entonces las herramientas CASE deben verificar la consistencia del modelo.

Las dificultades en la validación de requisitos no deben menospreciarse. Es difícil demostrar que un conjunto de requisitos cumple las necesidades del usuario. Los usuarios deben visualizar el sistema en operación e imaginarse la manera en que éste encajará en su trabajo. Para los profesionales de la computación es difícil llevar a cabo este tipo de análisis abstracto, pero para los usuarios del sistema es aún más difícil. Como resultado, la validación de requisitos claramente descubre todos los problemas en éstos por lo que es inevitable hacer cambios para corregir las omisiones y las malas interpretaciones después de que el documento de requisitos se ha aprobado.

4.7.1.- Revisión de requisitos

Éste es un proceso manual que involucra a varios lectores que verifican el documento de requisitos, tanto del personal del cliente como del contratista, en cuanto a anomalías y omisiones. Las revisiones de requisitos pueden ser informales o formales.

Las revisiones informales sencillamente implican que los contratistas deben discutir los requisitos con tantos stakeholders del sistema como sea posible. Antes de llevar a cabo una reunión para una revisión formal, se pueden detectar muchos problemas y errores en los requisitos simplemente hablando del sistema con los stakeholders.

En la revisión formal de requisitos, el equipo de desarrollo debe "conducir" al cliente a través de los requisitos del sistema, explicándole las implicaciones de cada requisito. El equipo de revisión debe verificar la consistencia de cada requisito y la integridad de estos como un todo. Los revisores también comprueban:

1. *Verificabilidad.* ¿El requisito puede probarse en la realidad?
2. *Comprensibilidad* ¿Los proveedores o usuario finales del sistema comprenden del todo el requisito?
3. *Rastreabilidad* ¿El origen de los requisitos está claramente establecido? Se tiene que ir hacia la fuente del requisito para evaluar el impacto del cambio. El rastreo es importante ya que permite evaluar el impacto del cambio en el resto del sistema. Esto se discute con mayor detalle en la siguiente sección.

4. *Adaptabilidad* ¿El requisito es adaptable: Es decir. ¿el requisito puede cambiarse sin causar efectos de gran escala en los otros requisitos del sistema"

Los conflictos, contradicciones, errores y omisiones en los requisitos deben señalarse durante la revisión y registrarse formalmente. Queda en los usuarios, el proveedor y el desarrollador del sistema negociar una solución para estos problemas identificados.

4.8.- Administración de requisitos

La ERS normalmente necesitará ser cambiada a medida que progresa el producto software. Es casi imposible especificar algunos detalles en el momento en el que se inicia el proyecto; por ejemplo, puede ser imposible definir durante la fase de requisitos todos los formatos de pantalla para un programa interactivo de forma que se garantice que no se modificarán más adelante. Además, es casi seguro que se realizarán cambios adicionales como consecuencia de haber encontrado deficiencias, defectos e inexactitudes que se descubren a medida que el producto evoluciona. Dos consideraciones a tener en cuenta en este proceso son:

1. El requisito debe ser especificado de la forma más completa posible, aun en el caso en que se prevean de forma inevitable revisiones en el proceso de desarrollo. Por ejemplo, los formatos de pantalla deseados deben especificarse lo mejor posible en la ERS, de forma que sirvan de base para el diseño posterior.
2. Debe iniciarse un proceso formal de cambio para identificar, controlar, seguir e informar de cambios proyectados tan pronto como sean identificados. Los cambios aprobados en los requisitos deben ser incluidos en la ERS de forma que permita:
   2.1. Suministrar una revisión precisa y completa del rastro de las modificaciones.
   2.2. Permitir un examen de fragmentos actuales y reemplazados de la ERS.

La administración de requisitos es el proceso de comprender y controlar los cambios en los requisitos del sistema. El proceso de administración de requisitos se lleva a cabo junto con los otros procesos de ingeniería de requisitos. La planificación comienza al mismo tiempo que la obtención de requisitos inicial y la administración activa de requisitos debe iniciar tan pronto como esté lista la primera versión del documento de requisitos.

4.8.1.- Requisitos duraderos y volátiles

Desde una perspectiva evolutiva, los requisitos son de dos clases:

1. *Requisitos duraderos* Éstos son relativamente estables que se derivan de la actividad principal de la organización y que están relacionados directamente con el dominio del sistema. Por ejemplo, en un hospital siempre habrá requisitos que se refieren a los pacientes, doctores, enfermeras, tratamientos, etcétera.
2. *Requisitos volátiles* Estos cambiarán probablemente durante el desarrollo del sistema o después de que éste se haya puesto en operación. Por ejemplo, por cambios de las políticas gubernamentales de salud.

Sommersville nos ofrece la siguiente división de los requisitos volátiles.

Requisitos mutantes	Requisitos que cambian debido a los cambios en el ambiente en el que opera la organización. Por ejemplo, en los sistemas hospitalarios, la consolidación del cuidado del paciente puede cambiar y requerir un tratamiento diferente de la información a recolectar.
Requisitos emergentes	Requisitos que emergen al incrementarse la comprensión del cliente en el desarrollo del sistema. El proceso de diseño puede revelar requisitos emergentes nuevos.
Requisitos consecutivos	Requisitos que son resultado de la introducción del sistema de cómputo. Esta introducción puede cambiar los procesos de la organización y abrir nuevas formas de trabajar que generarán nuevos requisitos del sistema.
Requisitos de compatibilidad	Requisitos que dependen de sistemas particulares o procesos de negocios dentro de la organización. Cuando estos últimos cambian, los requisitos de compatibilidad del sistema contratado o a entregar también pueden cambiar.

4.8.2.- Planificación de la administración de requisitos.

Ésta es una primera etapa esencial del progreso de administración de requisitos. La administración de requisitos es muy cara y, para cada proyecto, la etapa de planificación establece el nivel de detalle necesario en la administración de requisitos. Durante la etapa de administración de requisitos se tiene que decidir sobre:

1. *La identificación de requisitos* Cada requisito se debe identificar de forma única de tal forma que puedan entrar en referencia cruzada con otros requisitos de manera que pueda utilizarse en las evaluaciones de rastreo.
2. *Un proceso de administración del cambio* Éste es el conjunto de actividades que evalúa el impacto y costo de los cambios.
3. *Políticas de rastreo* Éstas definen la relación entre requisitos y la de éstos y el diseño del sistema que se debe registrar y la manera en que estos registros se deben mantener.
4. *Ayuda de herramientas CASE* La administración de requisitos comprende el procesamiento de grandes cantidades de información de los requisitos. Las herramientas que se pueden utilizar van desde sistemas de administración de requisitos especiales hasta hojas de cálculo y sistemas sencillos de bases de datos.

Existen relaciones entre los requisitos mismos y entre éstos y el diseño del sistema. También existen vínculos entre los requisitos y las razones de por qué éstos se propusieron. Cuando se proponen cambios se tiene que rastrear el impacto de estos cambios en los otros requisitos y el diseño del sistema. El rastreo es una propiedad de la especificación de requisitos que refleja la facilidad de encontrar requisitos relacionados.

Existen tres tipos de información de rastreo a las que se les debe dar mantenimiento:

1. *La información de rastreo de la fuente* vincula los requisitos con los *stakeholders* que propusieron los requisitos y la razón de éstos. Cuando un cambio es propuesto, esta información se utiliza para descubrir a los *stakeholders,* de forma que se les pueda consultar acerca de ese cambio.

2. *La información de rastreo de los requisitos* vincula los requisitos dependientes en el documento de requisitos. Esta información se utiliza para evaluar cómo muchos requisitos se ven probablemente afectados por un cambio propuesto y la magnitud de los cambios consecuentes en los requisitos.
3. *Información de rastreo del diseño* vincula los requisitos a los módulos del diseño en los cuales serán implementados. Esta información se utiliza para evaluar el impacto de los cambios de los requisitos propuestos en el diseño e implementación del sistema.

A menudo, la formación de rastreo implica utilizar matrices de rastreo que relacionan los requisitos con los *stakeholders*, entre los requisitos o entre los módulos del diseño. Si se consideran matrices de rastreo que vinculan requisitos con otros requisitos, cada uno de éstos se representa por una fila y una columna en la matriz. Donde existe una dependencia entre los requisitos ésta se registra en una celda en la intersección fila/columna.

Esto se ilustra en la figura que muestra una matriz de rastreo sencilla en la cual se registran las dependencias entre los requisitos. Una "U" en la intercesión fila/columna ilustra que el requisito en la fila utiliza los recursos especificados en el requisito señalado en la columna: una "R" significa que existe una relación débil entre los requisitos. Por ejemplo, pueden definirse requisitos para partes del mismo subsistema.

Req. Id.	**1.1**	**1.2**	**1.3**	**2.1**	**2.2**	**2.3**	**3.1**	**3.2**
1.1		U	R					
1.2			U			R		U
1.3	R			R				
2.1			R		U			U
2.2								U
2.3		R		U				
3.1								R
3.2							R	

Las matrices de rastreo se utilizan cuando se tiene que administrar un número pequeño de requisitos pero son muy pesadas y caras de mantener para sistemas grandes con muchos requisitos. Para estos sistemas, se tiene que capturar la información de rastreo en una base de datos de requisitos en la que cada requisito esté explícitamente vinculado a los requisitos relacionados. El impacto de los cambios se puede evaluar utilizando las facilidades de exploración de la base de datos. Alternativamente, es posible generar matrices de rastreo en forma automática.

La administración de requisitos necesita ayuda automática y las herramientas CASE a utilizarse deben elegirse durante la fase de planificación. Se requiere ayuda de las herramientas para:

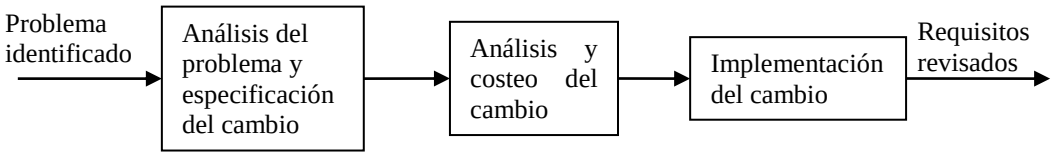
1. *Almacenar requisitos* Los requisitos deben mantenerse en un almacén de datos seguro y administrado que sea accesible a todos los que estén relacionados en el proceso de ingeniería de requisitos.
2. *Administrar el cambio* Este proceso se simplifica si está disponible una herramienta de ayuda.

3. *Administrar el rastreo* Como se discutió anteriormente. las herramientas de ayuda para el rastreo permiten relacionar los requisitos descubiertos. Para ayudar a descubrir las posibles relaciones entre los requisitos están disponibles algunas herramientas que utilizan técnicas de procesamiento de lenguaje natural.

Para sistemas pequeños, no es necesario utilizar herramientas de administración de requisitos especializadas. El proceso de administración de requisitos puede llevarse a cabo utilizando los recursos disponibles en los procesadores de texto, hojas de cálculo y bases de datos en PC. Sin embargo, para sistemas grandes, se requieren herramientas de ayuda más especializadas como DOORS, Requisite Pro o REM.

4.8.3.- Administración del cambio de requisitos.

Esta administración se aplica a todos los cambios propuestos en los requisitos. La ventaja de utilizar un proceso formal para administrar el cambio es que todos los cambios propuestos son tratados de forma consistente y que los cambios en el documento de requisitos se hacen de forma controlada.



Existen tres etapas principales en un proceso de administración de cambio:

1. *Análisis del problema y especificación del cambio* El proceso inicia con un problema de requisitos identificado o, algunas veces, con una propuesta de cambio específica. Durante esta etapa, el problema o la propuesta de cambio se analiza para verificar que ésta es válida. Entonces se hace una propuesta de cambio de requisitos más específica.
2. *Análisis del cambio y costeo* El efecto de un cambio propuesto se valora utilizando la información de rastreo y el conocimiento general de los requisitos del sistema. El costo de hacer un cambio se estima tanto en términos de modificaciones al documento de requisitos y, si es apropiado, como en el diseño e implementación del sistema. Una vez que el análisis se completa, se toma una decisión sobre si se procede o no en el cambio de requisitos.
3. *Implementación del cambio* Se modifica el documento de requisitos y. en su caso, el diseño e implementación del sistema. El documento se organiza para que los cambios puedan acomodarse sin tener que redactarlo todo nuevamente. Como con los programas de eventos, los cambios en los documentos se llevan a cabo minimizando las referencias externas y haciendo que las secciones del documento sean tan modulares como sea posible.

Si se requiere de forma urgente un cambio en los requisitos del sistema, existe siempre la tentación de hacer ese cambio al sistema y entonces modificar de forma retrospectiva el documento de requisitos. Esto inevitablemente conduce a que la especificación de requisitos y la implementación del sistema se desfasen. Una vez que se han hecho los cambios en el sistema, los del documento de requisitos se olvidan o se hacen de forma no consistente con los cambios del sistema.