Universidad Nacional de La Matanza

Departamento de Ingeniería e Investigaciones Tecnológicas

Asignatura: Ingeniería de Requerimientos

Notas de Clase:

Ingeniería de Requisitos del Software Orientada al Cliente

Capítulos 1, 2 y 3

Dra. Graciela D.S. Hadad

Revisión Marzo 2020

Índice

Acrónimos Utilizados

DEO	Discrepancias, Errores y Omisiones
EA	Escenarios Actuales
EF	Escenarios Futuros
ERS	Especificación de Requisitos del Software
IR	Ingeniería de Requisitos
LEL	Léxico Extendido del Lenguaje
LEL-S	Léxico Extendido del Lenguaje del Sistema
RF	Requisitos Funcionales
RNF	Requisitos No Funcionales
UdeD	Universo de Discurso

1. Requisitos del Software

Se presentan a continuación algunas definiciones de requisito, comenzando por la que provee la IEEE en el glosario de Ingeniería de Software [IEEE 24765].

Requisito del Software:

1. Una condición o capacidad que necesita un usuario para resolver un problema o alcanzar un objetivo.
2. Una condición o capacidad que debe cumplir o poseer un sistema o componente de sistema para satisfacer un contrato, estándar, especificación u otro documento formalmente impuesto.
3. Una representación documentada de una condición o capacidad como en 1 o 2.

IEEE glosario de Ingeniería de Software [IEEE 24765]

Requisitos:

Descripciones de lo que el sistema debe hacer - los servicios que presta y las limitaciones en su funcionamiento.

Ian Sommerville [Sommerville 11]

Requisito:

Una propiedad que debe ser exhibida por algo para resolver algún problema en el mundo real.

Software Engineering Body of Knowledge (SWEBoK) [Kotonya 14]

La definición de Software Engineering Body of Knowledge (SWEBoK) [Kotonya 14] se centra en el contexto del problema, no hace hincapié en el comportamiento del software sino en los efectos de dicho comportamiento sobre el contexto del problema. La definición de Sommerville [Sommerville 11] es la visión estricta del sistema. Analizando las dos primeras definiciones de [IEEE 24765], se observa que la primera es la visión de los clientes y usuarios, mientras que la segunda definición es la visión desde el sistema de software. La tercera definición dada en [IEEE 24765] corresponde a la forma en que se expresan dichos requisitos, denominado *especificación del requisito*.

Como se puede apreciar en la Figura 1, el proceso de construcción de requisitos tiene diversas entradas, siendo la salida los requisitos del software, cuya representación son las especificaciones.

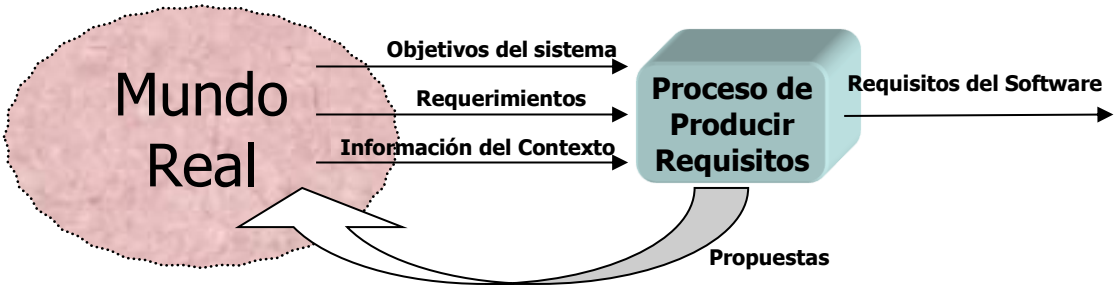

Figura 1. Requerimientos versus Requisitos

Con frecuencia se supone que todo pedido, demanda o necesidad presentada por los clientes y usuarios se traslada automáticamente al software. Sin embargo, la práctica real muestra que esta suposición suele ser falsa.

La aceptación incondicional de un pedido, demanda o necesidad por parte de los ingenieros de requisitos podría requerir soluciones tecnológicas inexistentes, presupuestos ilimitados, cronogramas inviables o crear dificultades políticas, legales o éticas. Es acá donde hay una clara distinción entre las demandas y lo que efectivamente un sistema de software podrá llevar a cabo, y es parte de las tareas de los ingenieros de requisitos realizar una transformación de los pedidos, demandas o necesidades (requerimientos) en las mejores condiciones factibles (requisitos) para el sistema de software a construir.

Debe tenerse en cuenta que el ingeniero de requisitos no solo debe capturar las demandas sobre lo que el software debe hacer sino también conocimiento necesario para entender lo que el software debe hacer. Es decir, debe también recolectar hechos del mundo real, lo que se denomina información del contexto.

Esta información proveniente del contexto es muy variada en propósito y granularidad, puede ser actual u obsoleta, relevante o fútil, y es el ingeniero de requisitos quien tamiza esa información en el proceso de construcción de requisitos en estrecha colaboración con los clientes y usuarios.

La información del mundo real no proviene exclusivamente de los clientes y usuarios, sino que existen otros tipos de fuentes de información, como documentos, sistemas de software, ambientes de trabajo/operación, otras

personas y grupos humanos, entre otras fuentes.

Cabe una aclaración respecto a las características del mundo real, los ingenieros de requisitos suelen tener la impresión de que están frente a un campo de verdad, pero se debe tener presente que casi inevitablemente esto no es así. Muchas veces la información transmitida desde ese mundo real puede ser incorrecta, ya sea por desconocimiento, obsolescencia o mala intención. Debe quedar bien claro que ese mundo real es imperfecto, dado lo cual, no debe atribuirse todas las falencias en los requisitos a errores de interpretación del ingeniero de requisitos.

De la Figura 1, se observa que las entradas al proceso de construcción de los requisitos son entonces:

- **Objetivos del sistema**: propuestos por los clientes; representan las características generales o propósito del sistema.
- **Información del Contexto**: proveniente de distintas fuentes de información, muy heterogénea en contenido, en granularidad, en vigencia y en utilidad. Básicamente se puede agrupar en:
  - Actividades actuales: provenientes principalmente de los usuarios; corresponden a descripciones de cómo se desarrollan las tareas, se operan dispositivos, se aplican resultados de cálculos u otros usos en general.
  - Propiedades o características: provenientes de distintas fuentes de información, como usuarios y documentación; corresponden, por ejemplo, a estándares de la organización, regulaciones gubernamentales, políticas de la organización, especificaciones técnicas de máquinas o de hardware.
- **Requerimientos:** provenientes de los usuarios y clientes; representan necesidades, deseos, imposiciones y limitaciones, que ellos expresan.
- **Propuestas:** elaboradas por los ingenieros de requisitos; representan básicamente ideas para mejorar las actividades en ese contexto a través del software o para facilitar el desarrollo del software manteniendo invariantes los objetivos. Cabe aclarar, que las propuestas no son siempre estrictamente entradas en sí mismas sino catalizadores de nuevas

entradas, posiblemente más coherentes con la visión del sistema de software. Las propuestas promueven una o varias situaciones de feedback donde los ingenieros de requisitos reciben rechazos, aceptaciones y aceptaciones parciales, lo que involucra un ciclo de negociación.

Los requerimientos son manifestados por los usuarios y clientes en diferentes niveles de abstracción. Si un usuario da instrucciones específicas sobre algo que el sistema de software deberá hacer, es muy probable que se convierta en un requisito del software, y la única consideración posible es determinar si es consistente o coherente con otros requisitos.

Por otro lado, cuando un requerimiento viene planteado en el formato de un problema actual a evitarse o de un deseo futuro poco preciso, el ingeniero de requisitos debe construir uno o más requisitos detallados que sinteticen ese pedido en situaciones o actividades en el contexto de aplicación, que posiblemente deban ser atendidas por el sistema de software.

En oposición a los requerimientos, el conocimiento acerca del contexto se debe capturar en una forma más detallada. Es así que se deben describir las actividades, operaciones o usos en ese contexto con un nivel de detalle tal que permita que luego sea posible planificar la inserción del sistema de software en forma armónica y ordenada. Estas descripciones deben procurar incluir información acerca de las razones que justifican estas actividades, operaciones o usos que se registran. Son justamente estas razones las que permitirán evaluar las diferentes opciones que surjan para insertar el sistema de software en ese contexto.

Las propuestas de los ingenieros de requisitos son una fuente de requisitos habitualmente poco considerada pero muy importante. Estas propuestas se confunden frecuentemente con actividades realizadas durante el Diseño de Software (principalmente cuando no hay una distinción clara entre el equipo de ingenieros de requisitos y el equipo de diseñadores) o con actividades correspondientes a la Reingeniería de los Procesos del Negocio o al Rediseño de Equipos (esto ocurre cuando el ingeniero de requisitos sugiere nuevas formas de hacer las cosas).

Se entiende entonces por propuestas de los ingenieros de requisitos a aquellas realizadas a los clientes y usuarios sobre cambios en un requerimiento o la incorporación de un nuevo requisito que intenta modificar en algún grado la forma en que el nuevo sistema brindará servicios o poseerá ciertas características diferentes. Por supuesto, estas propuestas deben ser explicadas, negociadas y documentadas.

Resumiendo, los términos requerimiento, requisito y especificación:

Requerimiento	necesidades, deseos, demandas y limitaciones provenientes de los clientes y usuarios
Requisito	condiciones y capacidades que debe cumplir el sistema de software, acordados entre los clientes, usuarios y desarrolladores
Especificación de requisito	representación de un requisito, realizada por los ingenieros de requisitos

Puntos importantes a tener en cuenta respecto a los requisitos:

- Los requisitos siempre existen.
- Los requisitos cambian.
- Los requisitos pueden manifestarse en forma implícita o explícita.
- Los requisitos se construyen.
- Los requisitos deben cumplir con los *objetivos del sistema* y estos, a su vez, deben estar alineados con los *objetivos estratégicos del cliente*.
- Los requisitos deben ser compatibles con el *contexto de uso*, considerando las habilidades y capacidades de adaptación de los usuarios.

1.1. Clasificación de Requisitos del Software

Una clasificación ampliamente difundida considera: requisitos funcionales (RF) y requisitos no funcionales (RNF). Esta clasificación sirve para simplificar su

presentación, hacerlos más legibles, entendibles y rastreables. A continuación, se presenta una definición de ambos tipos de requisitos:

Requisito Funcional	Servicios y actividades que debe proveer o realizar el sistema. Es decir, los RF expresan el comportamiento esperado del sistema.
Requisito No Funcional	Características y propiedades del sistema, así como también cualquier restricción que pueda limitar una solución. Es decir, los RNF expresan restricciones y cualidades que el sistema debe cumplir o poseer.

En resumen, se puede decir que los RF establecen qué debe hacer el sistema en términos de servicios a prestar, mientras que los RNF restringen cómo el sistema debe cumplir con esos servicios. Los servicios a prestar por el sistema pueden presentarse, por ejemplo, en la forma de entradas a procesar, funciones o procesos a realizar y salidas a proveer.

Se debe tener presente que un requisito no funcional no definido oportunamente puede obligar a muchos cambios en otros requisitos, tanto funcionales como no funcionales.

Algunos ejemplos de requisitos son:

- El sistema debe registrar cada cliente con los siguientes datos: nombre y apellido, domicilio, e-mail y teléfono. (RF)
- El sistema debe registrar un cliente nuevo en menos de 30 segundos. (RNF)
- El sistema debe registrar el alquiler de una película a un cliente, registrando al DVD como prestado. (RF)
- El sistema debe emitir un recibo para el cliente, en un máximo de 8 segundos después de registrar la operación de alquiler. (RNF)
- El sistema debe utilizar las técnicas de seguridad e integridad, indicadas en la normativa interna Nº 007-IIS/2007, sobre los datos almacenados. (RNF)
- El sistema debe realizar detecciones 10 veces por segundo a través del sensor óptico. (RNF)

1.2. Requisitos No Funcionales

Los RNF han sido objeto de numerosas clasificaciones, tanto en la literatura [Young 04] [Sommerville 11] como a través de normas internacionales [IEEE 29148] [ISO 25010]. La norma [ISO 25010] presenta un modelo de calidad para productos de software, donde define características de calidad con sub-características. De estas se presentan aquellas relacionadas con el sistema demandado. Entonces, usando la clasificación de esta norma se dispone de los siguientes tipos de RNF:

- **EFICIENCIA EN EL DESEMPEÑO**: conjunto de propiedades relacionadas con la capacidad del software de proveer un apropiado nivel de desempeño en relación a la cantidad de recursos necesarios bajo condiciones establecidas. Se define según las sub-características:
    - Comportamiento en el tiempo: tiempos de respuesta, tiempos de procesamiento y tasas de rendimiento del software.
    - Utilización de recursos: cantidades y tipos de recursos utilizados.
    - Capacidad: límites máximos de parámetros tales como número de elementos que se pueden almacenar, número de usuarios simultáneos, ancho de banda de comunicación, rendimiento de las transacciones y tamaño de la base de datos.

Ejemplos:

- *El sistema debe realizar la búsqueda de los datos del legajo del cliente en menos de 20 mseg.*
- *El sistema debe soportar 400 consultas por hora en promedio.*
- *El código ejecutable del sistema debe limitarse a 512 Kbytes.*
- *El software no debe ocupar más de un 20% de los recursos de procesamiento.*
- *El sistema debe eliminar el historial de reportes almacenados cuando este supere los 40 Mb.*

- **COMPATIBILIDAD**: conjunto de propiedades relacionadas con el intercambio de información y el desempeño con otro software, sistema o componente. Se

define según las sub-características:

- Co-Existencia: capacidad de desempeñarse eficientemente mientras comparte un entorno y recursos con otros productos, sin impactar negativamente en ellos.
- Interoperabilidad: capacidad de intercambiar información y utilizar la información intercambiada con otros productos.

Ejemplos:

- *El sistema debe recibir información del sistema de Facturación siguiendo el protocolo especificado en el apartado "Entradas Externas al Sistema".*
- *Los módulos del sistema se comunicarán mediante un archivo de intercambio.*
- *El sistema debe realizar las consultas a datos bancarios de clientes corporativos sobre una base de datos remota.*
- *El sistema deberá emitir los listados no impresos en formato CSV.*
- *El sistema debe almacenar las facturas en el controlador fiscal según la norma RG(AFIP) 1521.*

- **USABILIDAD**: conjunto de propiedades relacionadas con la capacidad del software para ser usado por determinados usuarios con efectividad, eficiencia y satisfacción en un contexto de uso específico. Se define según las sub-características:
  - Facilidad de reconocimiento: capacidad de brindar información de soporte apropiada al usuario, por ejemplo, mediante demostraciones, tutoriales o documentación.
  - Facilidad de aprendizaje: capacidad de ser utilizado con eficacia, eficiencia, ausencia de riesgo y satisfacción en un contexto específico.
  - Facilidad de uso: capacidad de ser operado y controlado por el usuario.
  - Protección contra errores de usuarios: capacidad del software para evitar que el usuario cometa errores.
  - Estética de interfaz de usuario: interfaz de usuario con una interacción agradable y satisfactoria para el usuario.
  - Accesibilidad: capacidad del software de ser usado por personas con distintas características y capacidades en un contexto de uso

específico.

Ejemplos:

- *Limitar a nivel 2 de navegación las principales funciones del sistema (ver detalle en apartado "Principales Funciones").*
- *Todas las opciones del sistema deben poder ejecutarse tanto con el teclado como con el mousse.*
- *El sistema debe poder ser operado por usuarios con conocimientos básicos de Windows y de Administración Contable.*
- *El sistema debe permitir seleccionar dos lenguajes para sus interfaces con usuarios: castellano e inglés.*

- **CONFIABILIDAD**: conjunto de propiedades relacionadas con la capacidad del software de mantener su nivel de prestación bajo condiciones establecidas durante un período de tiempo establecido. Se define según las sub-características:
    - Madurez: capacidad de ser confiable bajo una operatoria normal. Ejemplo de un parámetro de madurez es la frecuencia de falla.
    - Disponibilidad: capacidad de estar operativo y accesible cuando es requerido su uso. Ejemplo de un parámetro de disponibilidad es la proporción de tiempo total durante el cual el sistema está en estado operativo.
    - Tolerancia a fallas: capacidad de funcionar como es debido a pesar de la presencia de defectos de hardware o software.
    - Capacidad de recuperación: en caso de falla o interrupción, capacidad de recuperar los datos afectados y restablecer el estado deseado. Ejemplo de un parámetro de recuperación es el tiempo de inactividad después de cada falla.

Ejemplos:

- *El sistema debe tener una disponibilidad del 99%, es decir, satisfacer 99 pedidos sobre 100.*
- *El módulo de "transacciones seguras" debe implementar la característica de auto recuperación luego de alguna falla contemplada en el apartado 4 "Sobre fallas".*

- *El sistema on-line debe estar disponible las 24 horas del día de lunes a viernes.*
- *En caso de una falla en el sistema, éste debe seguir funcionando por lo menos en condiciones mínimas para resguardar los datos.*

- **SEGURIDAD**: conjunto de propiedades relacionadas con la protección de la información y los datos de manera que las personas u otros productos tengan el nivel de acceso apropiado según sus tipos y niveles de autorización. Se define según las sub-características:
    - Confidencialidad: capacidad de asegurar que los datos sean accesibles a las personas autorizadas a tener acceso.
    - Integridad: capacidad de prevenir el acceso o la modificación no autorizada de datos.
    - No repudio: capacidad de demostrar que las acciones tuvieron lugar, sin poder entonces ser rechazadas.
    - Responsabilidad: capacidad de identificar en forma unívoca a la entidad que generó las acciones.
    - Autenticidad: capacidad de probar la identidad de un sujeto o recurso.

Ejemplos:
- *El sistema debe encriptar toda comunicación externa usando el algoritmo RSA.*
- *Los reportes / archivos creados por un usuario deben estar ocultos para el resto de los usuarios.*
- *Solo los usuarios con privilegio de administrador podrán acceder a la totalidad de los datos.*
- *Las claves de los usuarios se encriptarán con un algoritmo propio de la empresa desarrolladora.*
- *El sistema debe utilizar conexiones seguras HTTP.*
- *Si la estación de trabajo no es utilizada, el sistema debe cerrar la sesión después de los 10 minutos de inactividad.*
- *El sistema debe registrar todas las operaciones en un log identificatorio del responsable de cada operación.*

- **MANTENIBILIDAD**: conjunto de propiedades relacionadas con la capacidad del software de ser modificado y probado. Las modificaciones pueden incluir correcciones, mejoras o adaptaciones del software. Se define según las sub-características:
    - Modularidad: capacidad de que un cambio en algún componente afecte mínimamente a otros componentes.
    - Reusabilidad: capacidad de los elementos constitutivos del software para ser reusables en otro software.
    - Facilidad de análisis: capacidad de evaluar cambios previstos o diagnosticar causas de falla. Por ejemplo, poseer mecanismos para analizar sus propios fallos o proveer informes antes de producirse un fallo.
    - Facilidad de cambio: capacidad de ser modificado sin introducir defectos o degradar su calidad. Es una combinación de variabilidad y estabilidad.
    - Facilidad de prueba: capacidad de ser probado según ciertos criterios.

Ejemplos:

- *El código fuente debe estar documentado en al menos un 40% del mismo.*
- *El código fuente de cada método debe tener un encabezado que describa su contenido y parámetros.*
- *Toda nueva versión del sistema debe poder instalarse remotamente.*
- *El sistema debe tomar los cambios de usuarios automáticamente del esquema de usuarios de la red de la organización.*
- *El sistema debe ser instalado por módulos, configurables según el perfil de usuario.*
- *El sistema debe soportar resguardos de datos programados por frecuencia o por volumen.*

- **PORTABILIDAD**: conjunto de propiedades relacionadas con la capacidad del software para ser transferido desde un entorno a otro. Se define según las sub-características:
    - Adaptabilidad: capacidad de ser adaptado para diferente hardware, software o entornos de uso, o diferentes versiones de estos. Incluye la

escalabilidad.

- Facilidad de instalación: capacidad de ser instalado y desinstalado con éxito en un entorno dado.
- Facilidad de reemplazo: capacidad de ser sustituido por otro software con el mismo propósito en el mismo entorno.

Ejemplos:

- *El sistema debe desarrollarse para plataformas de PC y Macintosh.*
- *El sistema debe funcionar tanto en Windows como en Linux.*
- *El módulo de impresión de comprobantes del sistema debe funcionar con las distintas marcas de impresoras fiscales disponibles a la fecha de entrega del sistema.*
- *El sistema debe funcionar en la plataforma Intel bajo la familia MS Windows.*

Es bien conocido el problema de escalabilidad del software, con lo cual se debe tener en consideración desde la Ingeniería de Requisitos el volumen de información a manejar, como también los tiempos y velocidades necesarias, dado que ello es determinante en la toma de decisiones posteriores sobre la arquitectura del software, la base de datos a utilizar u otros aspectos que atañen a su diseño. Es decir, estos atributos vinculados a un requisito deben describirse.

Ejemplos:

- Cantidad de facturas a emitir por día
- Cantidad de proveedores de la empresa
- Capacidad máxima de retención de agua por estanque
- Frecuencia de generación de datos estadísticos de venta
- Capacidad máxima de almacenamiento de datos por controlador móvil
- Promedio diario de pacientes ingresantes

2. La Importancia de los Requisitos en el Desarrollo del Software

La clave del éxito o fracaso de un proyecto de software depende fuertemente de atender las expectativas de los clientes, es decir, encontrar el conjunto de soluciones adecuadas para atender necesidades en el *universo de discurso* (UdeD).

> ***Universo de Discurso***: es el contexto general en el cual el software deberá ser desarrollado y deberá operar. Incluye todas las fuentes de información y todas las personas relacionadas con el software, denominados *stakeholders* o *involucrados*. Es la realidad acotada por el conjunto de objetivos establecidos por quienes demandan una solución de software. Las personas involucradas en el proceso de desarrollo son principalmente: usuarios, clientes, ingenieros de software, expertos del dominio.

Para lo cual, se debe primero capturar las demandas, necesidades y problemas existentes en el UdeD y luego determinar que los requisitos especificados sean los adecuados.

Varias décadas atrás, Frederick Brooks anunciaba en su artículo “No Silver Bullet” [Brooks 87]:

> *«The hardest single part of building a software system is deciding **precisely what to build**. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.»*

Los defectos en los requisitos incrementan altamente los costos de desarrollo y mantenimiento de los sistemas de software, dado que la corrección de los mismos puede involucrar desde simples adaptaciones al código, el rediseño de algún componente o hasta reformular total o parcialmente la solución propuesta. Los costos por corrección de defectos en los requisitos se incrementan en cada

fase de desarrollo del software. Barry Boehm [Boehm 81] estudió varios proyectos de desarrollo de software de gran envergadura para determinar el costo de corrección de defectos en los requisitos, dichos defectos eran descubiertos tardíamente en el proceso de desarrollo del software (ver Tabla 1).

Tabla 1. Costo relativo de corrección de un defecto [Boehm 81]

Fase donde se detectó el defecto	Costo promedio
Definición de Requisitos	1
Diseño	3 - 6
Codificación	10
Pruebas de desarrollo	15 - 40
Prueba de aceptación	30 - 70
Operación	40 - 100

Como se observa en la Tabla 1, el costo relativo de reparar un requisito erróneo que permanece sin ser detectado hasta la etapa de operación puede costar hasta 100 veces más que si fuera detectado y corregido en la fase de requisitos. En un estudio bastante posterior, Joseph Carr [Carr 00] mostró que la mayoría de estos números siguen siendo válidos.

Se puntualiza a continuación los aspectos a considerar en los defectos en los requisitos, según menciona Alan Davis [Davis 93]:

i) Cuanto más tarde en el ciclo de vida se detecta un defecto, más costoso es repararlo.

> ✓ *El crecimiento de los costos de reparación es producto de la catarata de defectos que se producen: defectos originados en una etapa se arrastran en fases sucesivas más nuevos defectos que se originan en cada una de ellas.*
>
> Modelo Mizuno (ver Figura 2)

En la Figura 2 se presenta el Modelo Mizuno [Mizuno 83] para las cataratas de defectos, el cual es notoriamente ilustrativo para comprender el problema de la propagación y la ampliación de los defectos a lo largo del proceso de desarrollo de software.

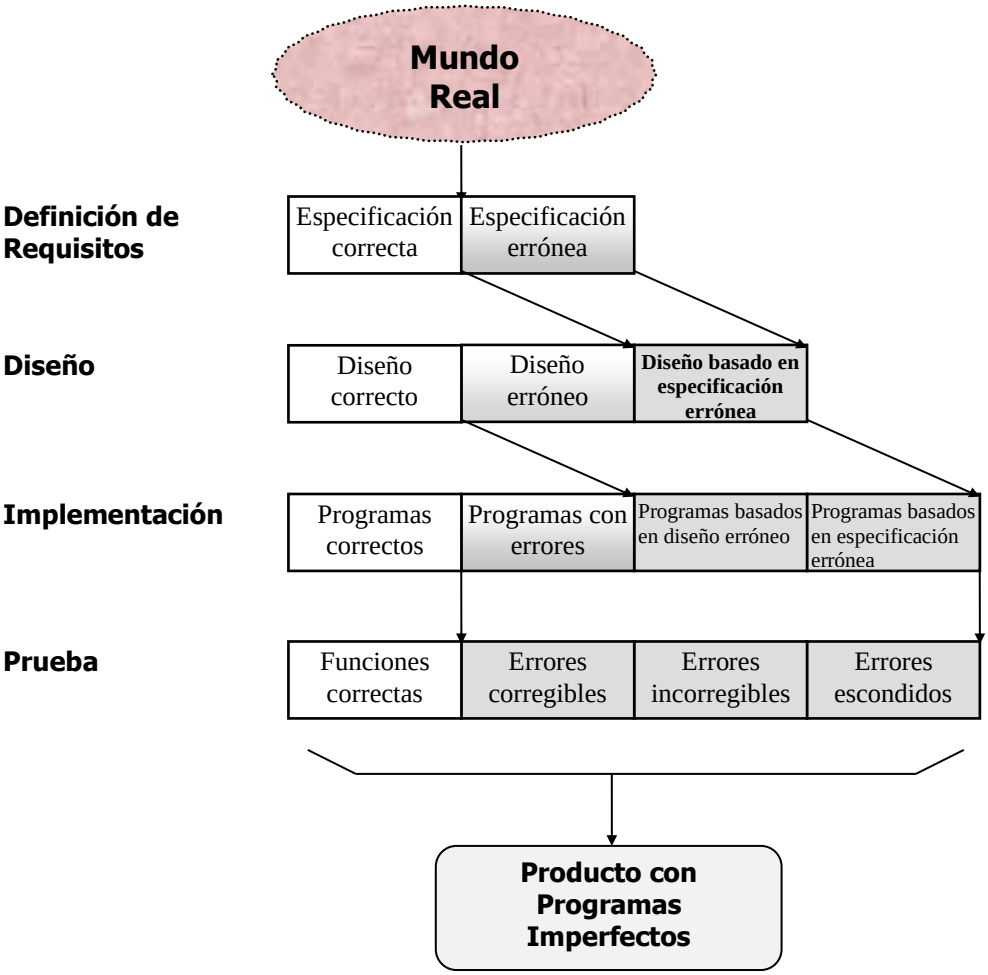

Figura 2. Catarata de Defectos según el Modelo Mizuno [Mizuno 83]

ii) Muchos defectos permanecen latentes y no son detectados hasta bastante después de la etapa en que se cometieron.

> ✓ *El 54% de los defectos se detectan después de la fase de codificación y prueba.*
> ✓ *Estos defectos provienen en un 45% de la fase de requisitos y en un 9% de la fase de codificación.*
>
> Barry Boehm [Boehm 75]

iii) Se están cometiendo demasiados defectos.

> ✓ *El 56% de los defectos tienen su origen en la etapa de requisitos.*
> Informado por Tom DeMarco, citado en Tavolato y Vincena [Tavolato 84]

iv) Los defectos de requisitos responden a una clasificación típica (ver Figura 3 con datos extraídos de [Basili 81]).

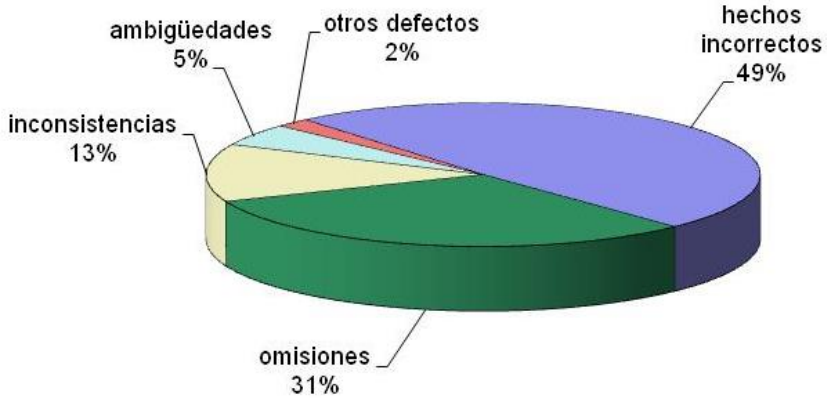

Figura 3. Tipos de Defectos en Requisitos [Basili 81]

v) Los defectos en los requisitos pueden detectarse.

> ✓ *La detección puede realizarse a través de distintas técnicas: ad hoc, inspecciones (muy alta efectividad), testeos (pruebas individuales), pruebas de integración, evaluación (pruebas de aceptación), otros.*
>
> ✓ *Las inspecciones detectan el 65% de defectos, el testeo de unidades: 10%, integración: 6%, evaluación: 10%, otros: 10%*
>
> Alan Davis [Davis 93]

Estos puntos se pueden resumir en las siguientes conclusiones [Davis 93]:

- Se cometen muchos errores de requisitos.
- Muchos de esos defectos se detectan tardíamente.
- Sin embargo, muchos de esos defectos pueden detectarse tempranamente.
- El no detectar defectos contribuye al incremento de costos en el software.

Diversas estadísticas realizadas a lo largo de las últimas décadas referidas a fracasos en proyectos de software [GAO 92] [Lutz 93] [CHAOS 95] concluyen que los requisitos son la fuente principal de problemas en la mayoría de dichos proyectos: requisitos inadecuados, cambios en los requisitos durante el ciclo de vida, requisitos no bien comprendidos, requisitos incompletos, etc.

En 1987, el Departamento de Defensa de los Estados Unidos generó un informe [Brooks 87b] sobre los problemas de calidad y productividad en el desarrollo de software militar, que abrió las puertas para la investigación de la llamada "*crisis*

del software", ya reconocida desde fines de la década del 60. Paralelamente, acaecieron en la práctica industrial fracasos paradigmáticos en proyectos de gran envergadura, tales como el Sistema de Control del Tráfico Aéreo de la Administración Federal de Aviación, el Sistema de Administración de Equipaje del Nuevo Aeropuerto Internacional de Denver, el Sistema de Ambulancias de Londres, el Sistema de tiempo real de Paramax System Corporation.

Algunos de los casos más recientes que se pueden mencionar son: el Sistema de Reservas del Ferrocarril Nacional francés (dos días fuera de uso por defectos en la actualización de un programa, 2004), el Sistema de Gestión de Energía de General Electric (causó un apagón en el noreste de Estados Unidos y parte de Canadá, debido a un error de programación en el sistema de gestión y monitoreo de energía de un proveedor, 2003), el software de la Estación Espacial Internacional (varios problemas como la operación de un brazo del robot de la estación, 2001), el programa de simulación del puente Millenium en Londres (falló debido a una incorrecta estimación de la cantidad de peatones, 2000), entre otros muchos.

Además de las consecuencias que producen los defectos en los requisitos, se deben analizar las causas de dichos defectos.

2.1. Evolución de los Defectos en el Proceso de Desarrollo de Software

Habitualmente cuando se hace referencia a defectos en los requisitos de los sistemas de software se desliza en forma tácita la idea de que el profesional de software o el proceso de captura han sido el origen de los mismos. Esta suposición, si bien muy difundida, probablemente sea parcialmente falsa ya que muchos de los errores, discrepancias y omisiones que aparecen en los requisitos y naturalmente se propagan al sistema final, ya existían en la fuente de información. En realidad, existe una responsabilidad compartida, ya que en los casos en que el defecto está empotrado en la fuente de información, el ingeniero de requisitos tiene algún grado de responsabilidad al no ponerlo en evidencia, especialmente en relación con las discrepancias y omisiones.

El Modelo Mizuno presentado en la Figura 2, si bien facilita comprender el problema de la propagación de los defectos a lo largo del proceso de desarrollo de software, contiene varias simplificaciones que enmascaran algunos aspectos relevantes de la problemática de la Ingeniería de Requisitos.

El punto más importante parece ser la visión del *problema real* como una entidad fuente de todo saber y verdad. No existe un *problema real* perfecto que determine unívocamente los requisitos del sistema de software. Por el contrario, el *problema real* reside en que las organizaciones no son completas ni coherentes [Godel 62]. Además, las fuentes de información introducen sus propias contradicciones, sesgos, errores y extemporalidades que obscurecen un poco más la percepción de las necesidades reales en ese UdeD.

El modelo conceptual que supone un "*problema real*" perfecto, asigna además todas las dificultades a:

i) la inhabilidad de las fuentes de información para revelar el conocimiento profundo, y

ii) la inhabilidad del ingeniero de requisitos para capturar la información relevante y sin omisiones.

El "*problema real*" es un agregado de certezas, contradicciones, dudas y aspectos no planteados. En otras palabras, sí hay información directamente transformable en requisitos del sistema de software, pero hay mucho más que eso. La tarea del ingeniero de requisitos es mucho más compleja que meramente sonsacar los requisitos, también debe colaborar con el esclarecimiento de las dudas, debe detectar las contradicciones y visualizar las circunstancias no consideradas, entre otras. Adicionalmente, el ingeniero de requisitos contribuye con propuestas de servicios y características del sistema, que en algunos casos complementan la información adquirida y que en algunos otros ofrece algún antagonismo con la misma.

Los requerimientos, pedidos y demandas de los clientes y usuarios son idealmente realizados con el objetivo de obtener una mejora en los procesos del negocio. Se dice "idealmente" porque pueden insertarse intereses personales no claramente explicitados o eventualmente cuidadosamente ocultos. Por otro lado, las propuestas del ingeniero de requisitos están guiadas por la aspiración de

facilitar el desarrollo del artefacto de software, ya sea simplificando el sistema o dotándolo de propiedades que lo acerquen a modelos conceptuales preconcebidos.

En la Figura 4, se presenta un esquema que extiende el modelo de Mizuno, visualizando el *problema real* como un agregado de certezas, contradicciones, dudas y aspectos no considerados. En este esquema además se representa el hecho de que las fuentes de información aportan su propia distorsión dificultando, en alguna medida, aún más el proceso de definición de los requisitos del sistema de software.

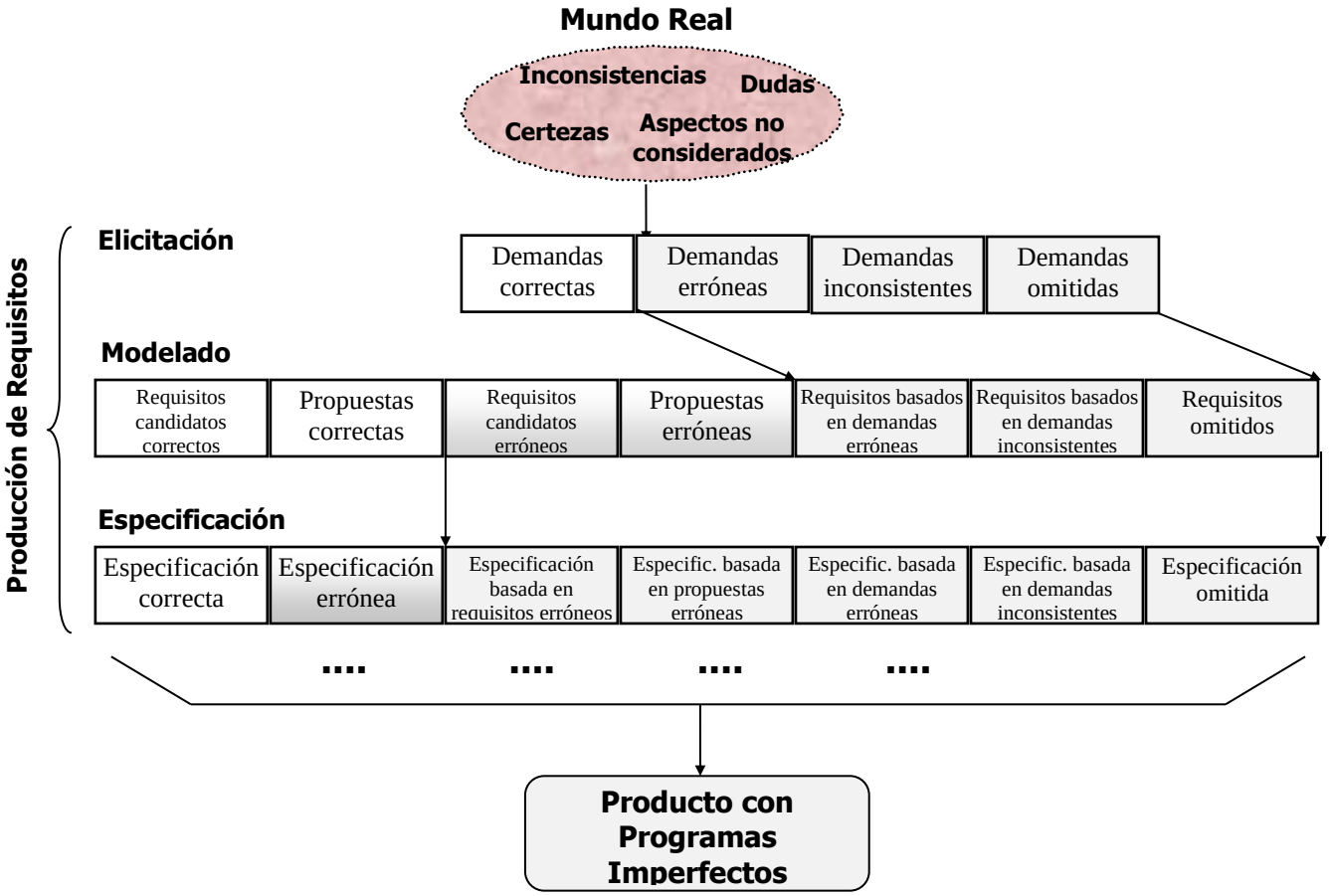

Figura 4. Catarata de Defectos extendiendo el Modelo Mizuno[1]

La Figura 4 fue confeccionada siguiendo el estilo de la figura original de Mizuno, por lo que conserva las mismas simplificaciones que la original, excepto las ya indicadas modificaciones en la visión del problema real. En ese sentido, el esquema no refleja que sobre los requisitos erróneos se puedan además

[1] Por razones de espacio, solo se muestra la fase "Definición de Requisitos", abierta en sus actividades.

acumular defectos de diseño (sólo se comenten defectos de diseño sobre los requisitos correctos) y que los errores de programación se concentran en los fragmentos del sistema correctamente diseñados. Por otra parte, en ambas figuras también se ignora que una actividad de validación forzaría, con un alto costo, la corrección de los “defectos incorregibles”; así como descubriría en mayor o menor grado los “defectos escondidos”, también con altos costos de corrección.

2.2. Evolución de los Requisitos

La Ingeniería de Software desde sus comienzos ha propuesto métodos para facilitar los cambios en el software, desde los métodos estructurados, ocultando información en módulos hasta los métodos orientados a objetos encapsulando comportamiento en clases. Los cambios no son ocasionales, sino que muy por el contrario ocurren constantemente durante todo el ciclo de vida del software. Como bien menciona Julio Leite [Leite 97]: «*el cambio es una propiedad intrínseca al software*», y por lo tanto merece una consideración especial en el proceso de desarrollo, que es la administración de dichos cambios. Desde la perspectiva de la Ingeniería de Requisitos, esta actividad se denomina “*gestión de requisitos*”.

Los requisitos evolucionan registrando los cambios que el UdeD impone al software. Estadísticas realizadas marcan que el 50% o más de los requisitos van a cambiar antes que el sistema de software se ponga en operación [Kotonya 98]. Es por esta razón que se pone énfasis en especificaciones dinámicas de requisitos precisos y claros que acompañen los cambios del UdeD y que permitan el seguimiento de los mismos desde cualquier punto del proceso de desarrollo del software hasta sus orígenes.

Debe considerarse cuál es el origen de estos cambios en el software. *¿Son estos cambios realmente genuinos?*, es decir, *¿surgen como consecuencia de cambios en el UdeD o provienen de requisitos descubiertos tardíamente?*

Es frecuente que algunos requisitos se descubran parcialmente o no salgan a la luz hasta avanzado el desarrollo, cuando los diseñadores o inclusive los

programadores deben realizar tareas a muy bajo nivel de detalle y se chocan con ambigüedades, contradicciones u omisiones. En estos casos, la esencia del cambio no corresponde a un mundo en evolución sino a requisitos no descubiertos oportunamente.

Con respecto a los cambios “reales” en el UdeD, no sólo se deben a cambios externos o internos a la organización como consecuencia de factores políticos, económicos, legales, sociales o tecnológicos, sino muchas veces a un cambio en las expectativas de los mismos clientes y usuarios una vez que ven más claramente las posibilidades que permitirá el sistema de software. Lehman [Lehman 80] va más allá expresando que un sistema una vez instalado se convierte en parte del UdeD alterándolo y, por ende, él mismo (el software) altera sus propios requisitos.

En general, las estadísticas realizadas sobre la evolución de los requisitos no discriminan adecuadamente el origen de los cambios en el desarrollo del software. Lientz y Swanson [Lientz 78] determinaron que los cambios en el software durante su mantenimiento se debían en un 80% a cambios en los requisitos y el restante 20% a correcciones de defectos. Sin embargo, no se discrimina en estos valores cuántas de las correcciones se debían a requisitos con defectos ni cuántos de los cambios en los requisitos eran por la naturaleza evolutiva del UdeD o por defectos en los requisitos (requisitos descubiertos tardíamente o requisitos mal interpretados). Un estudio posterior sobre mantenimiento de software [Pigoski 96] informa que alrededor del 55% de los pedidos de cambio corresponden a requisitos nuevos o cambios en requisitos por mejoras, mientras que un 25% son cambios técnicos por adaptaciones.

Desde ya que existen universos de discurso donde predomina una evolución constante, tal es el caso por ejemplo del dominio de las telecomunicaciones. También es el caso de desarrollos de sistemas grandes, complejos donde es inevitable que haya múltiples versiones que derivan tanto de cambios planeados por mejoras como de cambios no planeados por evolución del UdeD. Por otro lado, el descubrimiento tardío de requisitos se debe en muchos casos a procesos de construcción “débiles” con pocas guías y poca disciplina, donde se requiere la generación rápida del artefacto de software y los requisitos se descubren a

medida que se implementa parcialmente el software, tal es el caso en el dominio del comercio electrónico donde hay una fuerte competencia. Es también muy común que el descubrimiento tardío corresponda a requisitos no funcionales, cuando éstos no son atendidos adecuadamente durante la definición de requisitos [Cysneiros 99]. Un requisito no funcional no capturado oportunamente puede obligar a muchos cambios en otros requisitos tanto funcionales como no funcionales.

De lo mencionado hasta ahora, cabe destacar dos cuestiones:

i) La visión del Modelo de Mizuno (Figura 2) y la visión extendida del mismo (Figura 4) son estáticas. Ambas consideran que los procesos del negocio no cambian durante el período de desarrollo del software.

ii) La visión muy difundida de la evolución de los requisitos se basa en un único componente: el cambio del UdeD, pero prácticamente omite un segundo componente: los requisitos descubiertos o comprendidos tardíamente. Es obvio que este último componente produce también modificaciones en los requisitos, y lo hace con mucha mayor frecuencia de lo mencionado en la literatura.

Jackson [Jackson 94] afirma que sobre los requisitos se saben dos cosas:

i) van a cambiar, y

ii) van a ser mal comprendidos.

En la Figura 5 se muestra la incorporación de la noción de evolución de los requisitos a la Figura 4, donde se presenta un eje relativo al tiempo de desarrollo del software y otro relativo al tiempo de los cambios durante el desarrollo mismo.

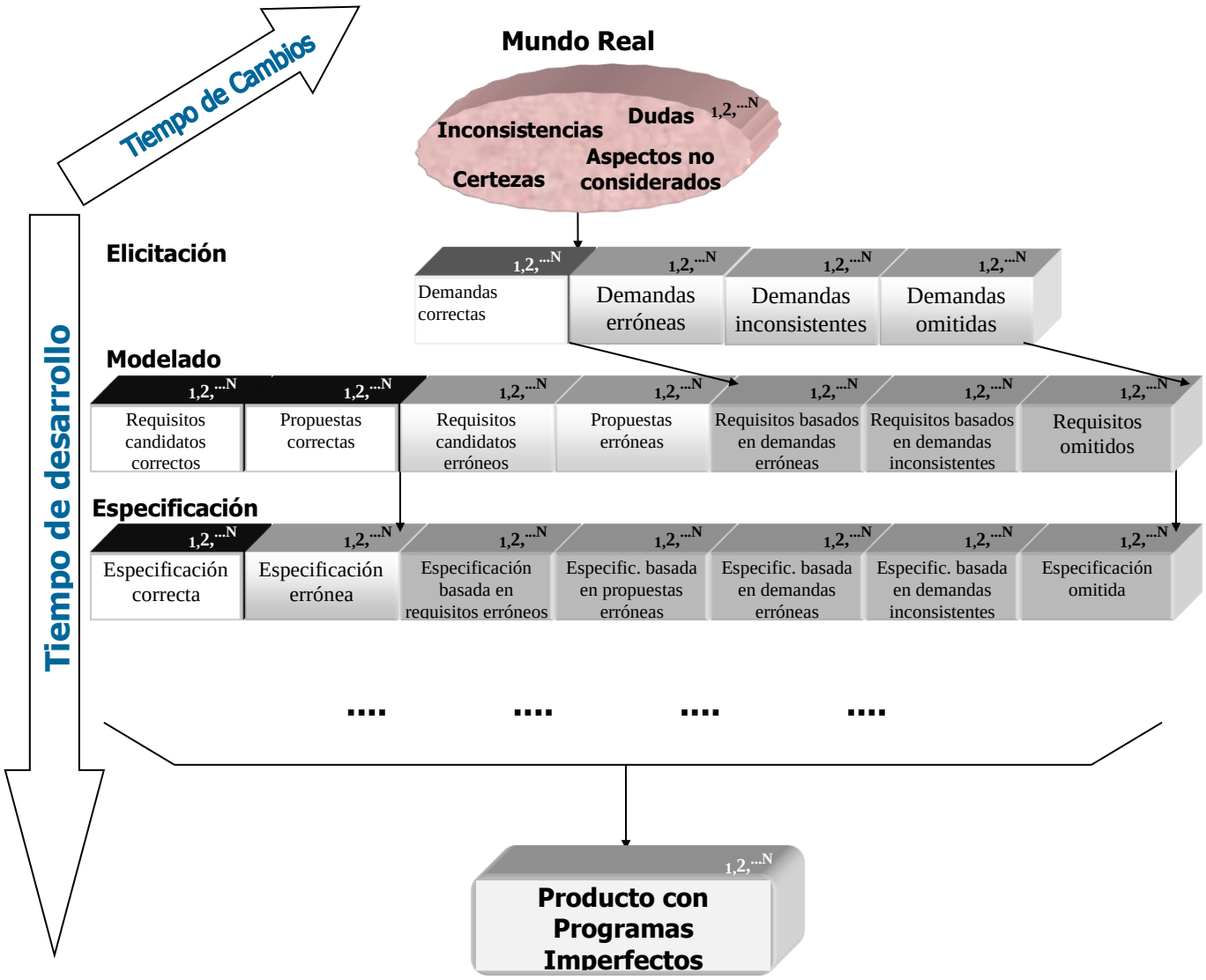

Figura 5. Evolución de los Defectos + Evolución de los Requisitos

El proceso de construcción de Requisitos tiene sentido en organizaciones con una estabilidad aceptable en sus procesos del negocio. En organizaciones con estrategias o procesos del negocio inestables a muy inestables, hay que pensar en métodos ágiles de desarrollo de software, pues cuál es el sentido de modelar requisitos que cambian más rápido que el tiempo necesario para documentarlos.

3. La Ingeniería de Requisitos en el Contexto de la Ingeniería de Software

La Ingeniería de Requisitos (IR) es una disciplina que procura sistematizar el proceso de construcción de requisitos. Es decir, apunta a mejorar la forma en que se comprenden y definen sistemas de software complejos. La complejidad de los sistemas exige que se preste más atención al correcto entendimiento del problema antes de comprometer una solución. Un proceso de IR cubre todas las actividades involucradas en descubrir, documentar, analizar y mantener un conjunto de requisitos para un sistema de software.

Aunque el problema de cómo establecer sistemas no es nuevo, sólo en las últimas dos décadas la comunidad científica de computación, principalmente la comunidad de Ingeniería de Software, comenzó a prestar especial atención a la tarea de definición de requisitos. El término "*Ingeniería de Requisitos*" ("*Requirements Engineering*") surgió formalmente publicado en enero de 1977 en IEEE Transactions on Software Engineering. La primera conferencia internacional sobre este tema fue en 1993 (RE'93) y en 1995 se realizó la primera reunión del grupo de trabajo IFIP WG2.9 – Software Requirements Engineering (Internacional Federation for Information Processing Working Group 2.9).

La Ingeniería de Software ha estado prestando demasiada atención al modelado e implementación de productos, pero la IR comenzó examinando los aspectos de la generación de requisitos y su intrínseca dependencia de los aspectos sociales que rodean el desarrollo del software y que rodearán su operación. La IR tiene una interacción muy fuerte con aquellos que demandan un producto de software. Por lo tanto, el proceso de construcción de requisitos es un esfuerzo cooperativo que involucra a usuarios, clientes, ingenieros de software, consultores y expertos, entre otros, denominados *involucrados* en la literatura (traducción del término inglés *stakeholders*).

La IR debe entonces achicar la brecha entre los usuarios que tienen necesidades y requieren soluciones a sus problemas, pero que no tienen los conocimientos para especificar los requisitos, y los ingenieros de requisitos que sí tienen dichos

conocimientos técnicos, pero desconocen el mundo de los usuarios con sus problemas y necesidades. De lo dicho, surgen dos aspectos ineludibles en un proceso de definición de requisitos exitoso:

i) garantizar una buena comunicación entre ambas partes, y
ii) comprender cabalmente el mundo del usuario.

Lo primero se logra cuando los ingenieros de requisitos comprenden el vocabulario que manejan los usuarios. Para lo segundo, los ingenieros de requisitos requieren utilizar técnicas específicas que ayuden a obtener información de ese mundo (técnicas de elicitación[2]) y que faciliten determinar si la comprensión lograda es correcta (técnicas de verificación y validación). Cabe destacar que cuando la fuente de información son las personas, éstas muchas veces no saben o no pueden o no quieren expresar claramente sus necesidades, o lo que es peor aún, pueden ni siquiera ser conscientes de ellas.

En resumen, la necesidad de asegurar una buena comprensión entre los ingenieros de requisitos y los usuarios (en general, entre todos los involucrados) ha inducido al desarrollo de métodos que permitan la colaboración entre todos los participantes del proceso de definición de requisitos. Los ingenieros de requisitos deben comprender, modelar y analizar el UdeD donde el software operará y los usuarios deben confirmar que la visión de los ingenieros es correcta. Estos requisitos deben contemplar las necesidades de los usuarios, las reglas de la organización y las reglas externas próximas al UdeD.

Debido a la baja calidad de los productos de software con altos costos de corrección y mantenimiento, los desarrolladores de software e investigadores han profundizado sus estudios y prácticas en el proceso previo al diseño del software. Se debe garantizar la producción de software de alta calidad y bajo costo estableciendo un punto de partida de alta confiabilidad, el cual se logra mediante la detección lo más tempranamente posible de defectos. Es necesario entonces contar con procesos de construcción de requisitos confiables.

Por otra parte, la IR enfatiza la necesidad de que los requisitos no sólo sean de la mayor calidad posible en el comienzo del proceso de desarrollo, sino que

[2] **Elicitar**: descubrir, tornar explícito, obtener el máximo de información para el conocimiento del objeto en cuestión.

también se conserve la calidad a lo largo de todo el desarrollo. Para lograr esto, es necesario que los requisitos puedan evolucionar registrando los cambios que la realidad impone al desarrollo de la aplicación. Por esta razón, se pone énfasis en especificaciones dinámicas de requisitos precisos y claros que acompañen los cambios del UdeD y que permitan el seguimiento de los mismos desde cualquier punto del proceso de desarrollo del software hasta sus orígenes. Se puede decir que la sistematización del proceso de detección de defectos ayuda considerablemente en el proceso de especificación de requisitos, y esto se refleja en todo el proceso de desarrollo del software.

3.1. El Proceso de Ingeniería de Requisitos

La IR establece el proceso de construcción de requisitos como un proceso en el cual lo que debe ser hecho es elicitado, modelado y analizado. Este proceso debe lidiar con distintos puntos de vista y usar una combinación de métodos, herramientas, procedimientos y personal. El producto final de este proceso es un documento denominado *Especificación de los Requisitos de Software* (ERS), que puede estar acompañado de otros documentos y modelos, pero el objetivo de este proceso es obtener una comprensión acabada del problema de nuestros clientes para establecer la solución más adecuada al problema.

Este proceso ocurre en un contexto previamente definido: el UdeD, el cual permanece en constante evolución durante el proceso de desarrollo y mantenimiento del software. Debido a lo cual, surgen nuevos requisitos y cambian requisitos existentes. Por ende, la IR debe atender estos cambios durante todo el ciclo de vida del sistema de software.

El proceso de construcción de requisitos, a veces denominado proceso de definición de requisitos, es naturalmente iterativo e involucra las siguientes actividades: la elicitación, el modelado y el análisis de los requisitos, y una cuarta actividad cruzada que es la gestión de requisitos (ver Tabla 2).

Tabla 2. Actividades del Proceso de Ingeniería de Requisitos

Elicitación	Modelado	Análisis	Gestión
▪ Identificación de Fuentes de Información ▪ Recolección de hechos ▪ Comunicación	▪ Representación ▪ Organización ▪ Almacenamiento	▪ Verificación ▪ Validación ▪ Negociación	▪ Identificación de cambios ▪ Análisis de cambios ▪ Realización de cambios

Estas actividades interactúan entre sí desarrollándose concurrentemente, por ejemplo, a medida que se va adquiriendo conocimiento, se va modelando y analizando la información recolectada; mientras que la gestión de requisitos también involucra a estas tres actividades a lo largo del tiempo acompañando la evolución del UdeD.

La Figura 6 muestra la iteración entre estas actividades por retroalimentación junto con la evolución del UdeD. Es decir, el proceso de construcción de requisitos continúa a lo largo de todo el ciclo de vida del software.

Con el fin de definir los requisitos del sistema de software, se requiere un profundo conocimiento del UdeD. Esto incluye no sólo conocer la operatoria actual, sino también los planes para el futuro. A menos que el ingeniero de requisitos tenga un conocimiento previo del dominio, este puede sobrecargarse con datos de diferente naturaleza durante la adquisición de conocimiento. Para hacer frente a esto, es aconsejable aplicar un proceso gradual que migre lentamente del aprendizaje a la definición. Durante la fase de aprendizaje, la atención debe centrarse en los hechos actuales, que deben ser organizados y registrados. Más tarde, deben entenderse los planes hacia el futuro, pero en lugar de registrarse tal como se presentan, deben ser utilizados para diseñar los modelos del futuro UdeD. Tanto en la fase de aprendizaje como en la de definición, se elicita, modela y analiza, aunque distinto tipo de información: inicialmente actual y luego futura, y paralelamente se desenvuelve la gestión de requisitos, actividad cruzada que está presente a lo largo de todo el proceso de desarrollo y mantenimiento del software.

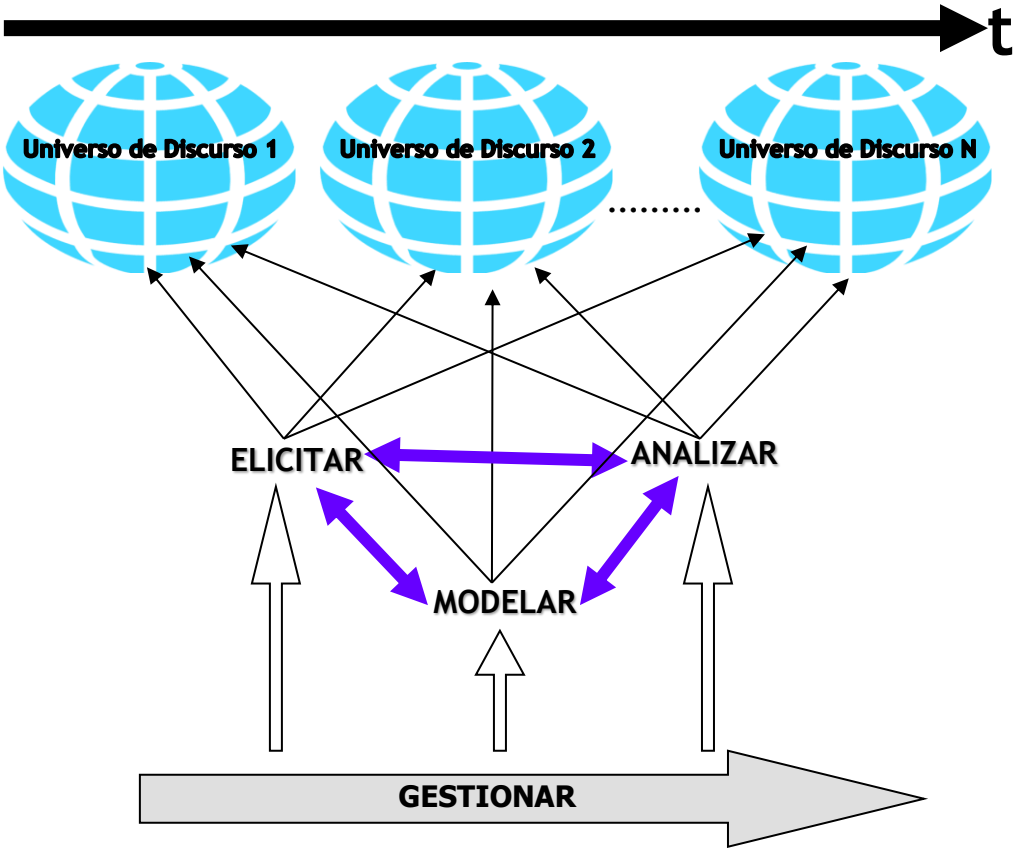

Figura 6. Evolución del universo de discurso frente a las actividades del proceso de requisitos

Cabe señalar que, en circunstancias específicas, tales como un profundo conocimiento previo del UdeD por parte del equipo de desarrollo, ese aprendizaje sobre el mundo real puede reducirse e incluso anularse. Para ello, deben existir mecanismos que permitan asegurar que no es necesaria esa etapa de aprendizaje.

3.2. Actividades del Proceso de Ingeniería de Requisitos

La salida del proceso es un conjunto de requisitos definidos, verificados, validados y acordados. Este conjunto de requisitos suele estar plasmado en el documento ERS, el cual puede estar acompañado de modelos. Algunos de estos modelos suelen estar dedicados a representar el contexto en el que el software se va a desempeñar y la contribución de estos modelos es agregar semántica al

ERS.

Se debe considerar que el proceso de IR, como todo proceso, transforma entradas en salidas. Las entradas corresponden a información contenida dentro de ese UdeD cambiante, heterogéneo, disperso e imperfecto. Esa información posee diversas características contrapuestas: probablemente desorganizada, relevante e irrelevante, con distinta granularidad, correcta e incorrecta, coherente y contradictoria, abierta / explícita y oculta / implícita, directamente entendible y requerir interpretación / traducción, vigente y desactualizada / obsoleta, proveniente de diversas fuentes de información, distintos medios de soporte, entre otros aspectos. Mientras que se espera que las salidas se correspondan a información organizada, consistente, lo más completa posible, correcta, sin ambigüedades, cuyo medio de soporte es habitualmente el documento ERS y los modelos que describen el ambiente y el sistema desde distintas perspectivas y para distintos destinatarios. De lo dicho se desprende que, la transformación de las entradas heterogéneas en estas salidas es una tarea ardua que requiere tiempo, personal adiestrado, métodos y técnicas adecuadas y herramientas de apoyo.

Por otro lado, cabe destacar que el proceso de requisitos es un proceso infinito que está siempre ocurriendo mientras el software evoluciona, dado lo cual la salida del proceso no es única, sino que también irá evolucionando continuamente, escapando del *congelamiento de los requisitos*.

A continuación, se describen brevemente las actividades del proceso de requisitos.

Elicitación

Se encarga de encontrar los hechos relevantes del UdeD. Es decir, es una fase de adquisición de conocimiento. Primeramente, se identifican y priorizan las fuentes de información, se seleccionan las técnicas de elicitación más adecuadas al problema en cuestión y luego, comienza la recolección de hechos propiamente dicha. En esta fase es crucial la comunicación que se establece entre los involucrados.

Modelado

Se encarga de representar, organizar y registrar (almacenar) los hechos recolectados durante la elicitación. Es decir, éstos deben documentarse en una forma organizada, comprensible y significativa. El modelo puede estar compuesto de distintos conjuntos de representaciones. Se han adaptado muchos modelos del diseño para modelar requisitos, mientras que se han creado modelos a partir de las necesidades propias de la fase de definición de requisitos. Varios de estos modelos se basan en el uso del lenguaje natural, tales como los Casos de Uso [Cockburn 00], los Escenarios [Potts 94], el Léxico Extendido del Lenguaje [Leite 93] y las Historias de Usuario [Cohn 04], entre otros.

Análisis

Debe verificar y validar los hechos modelados. Se encarga de descubrir y resolver los problemas que se presentan con los requisitos, por ejemplo: requisitos omitidos, contradictorios, no realizables, superpuestos, ambiguos, erróneos, aplicando diversas técnicas de verificación y validación. Asimismo, esta actividad involucra la negociación de propuestas de requisitos elaboradas por los ingenieros de requisitos y la negociación por conflictos de intereses entre los involucrados. La negociación involucra entonces el intercambio de información, la discusión, la resolución de conflictos, el acuerdo de propuestas con los clientes y usuarios, y la asignación de prioridades a los requisitos.

Gestión de Requisitos

Debe identificar, analizar y realizar los cambios necesarios provenientes de nuevos requisitos o modificaciones a requisitos existentes. Esta actividad se encarga no sólo de identificar nuevos requisitos o cambios en requisitos, sino también identificar los requisitos afectados por estos cambios y estimar los costos del cambio. Una vez efectivizados los cambios en los documentos y modelos de requisitos, estos cambios a su vez deben verificarse y validarse. Es decir, en esta actividad se realizan las tres actividades previamente mencionadas. Dos actividades indispensables en la gestión de los requisitos son: la rastreabilidad de los requisitos y el control del versionado, debido a la evolución de los requisitos. La rastreabilidad de los requisitos (*requirements*

traceability) se encarga de mantener vínculos entre los requisitos, el diseño y el código, y entre los requisitos y las fuentes de información que les dieron origen.

3.3. Puntos Clave en el Proceso de Requisitos

En base a lo ya mencionado hasta aquí, los puntos clave que debe considerar un proceso de definición de requisitos son:

i) **Involucrar a los clientes y usuarios** en el proceso. Es decir, establecer la participación activa y constante de los clientes y usuarios. Esto implica entablar una buena comunicación con ellos para comprender cabalmente sus necesidades.

ii) Asegurar la **calidad de los requisitos**. Esto involucra utilizar técnicas adecuadas de Verificación y Validación, las cuales deben formar parte de las actividades continuas del proceso de definición de requisitos y no como hitos estáticos. Por otro lado, la calidad de los requisitos debe conservarse a lo largo de todo el ciclo de vida del software.

iii) Considerar la **evolución de los requisitos** durante el ciclo de vida del software. El proceso debe acompañar esta evolución. Esto involucra una adecuada gestión de los requisitos, asegurando su rastreabilidad.

iv) Considerar **aspectos sociales, organizacionales, políticos y legales del ambiente** donde el software operará. Esto impone una inmersión de los desarrolladores en dicho ambiente y el uso de técnicas de modelado organizacional.

v) Procurar la **completitud de los requisitos**. Esto es por un lado, un aspecto de calidad y, por otro lado, requiere de una comprensión adecuada del UdeD, ambos aspectos tratados en los puntos anteriores.

Dados los graves problemas que los defectos en los requisitos ocasionan sobre el producto final (el sistema de software) y durante el proceso de construcción del mismo, es que diversos centros de estudio, organismos gubernamentales y otras organizaciones se han dedicado en la última década a la investigación en el área de los requisitos de los sistemas de software.

En tal sentido, se han propuesto diversos enfoques en el tratamiento de requisitos, estableciendo nuevos procesos, técnicas y métodos de construcción de requisitos, creando herramientas de soporte a dichos procesos, fijando estándares en los procesos y productos, y estableciendo procesos para garantizar la calidad de los requisitos, procesos que permitan la evolución de los requisitos y su adecuada gestión durante todo el ciclo de vida del software.