Universidad Nacional de La Matanza

SECRETARÍA DEPARTAMENTAL DE INVESTIGACIONES - DIIT

SDINV - DIIT - NOTA : 56 - 2025
FECHA DE EMISION: 14/04/2025

SRA. ACADÉMICA:

Por medio de la presente, se solicita subir al repositorio digital de la Biblioteca de la UNLaM el libro titulado "Ingeniería de Requisitos", de la autora Dra. Gladys Kaplan.

Saludos cordiales.

A LA SECRETARIA ACADEMICA
MG. ANA BIDIÑA
S _____ /_____ D

Gladys N. Kaplan

INGENIERÍA DE REQUISITOS

El camino hacia un software de calidad comienza en un punto de partida confiable

DIIT
Departamento de Ingeniería e Investigaciones Tecnológicas
UNLaM
1989

Autoridades UNLaM

Rector/
Prof. Dr. Daniel Eduardo Martínez

Vicerrector/
Dr. Fernando Luján Acosta

Vicerrector Ejecutivo/
Mag. Gustavo Duek

Secretario General/
Lic. Sebastián Garber

Secretaria Académica/
Mag. Ana Bidiña

Secretario de Ciencia y Tecnología/
Lic. Juan Pablo Piñeiro

Secretario de Extensión Universitaria/
Lic. Roberto Luis Ayub

Secretario de Desarrollo Universitario/
Lic. Nicolás Martínez

Secretario Administrativo/
Cdor. Leonardo Minoli

Secretario de Informática y Comunicaciones/
Mag. Ing. Osvaldo Mario Spositto

Secretario Legal y Técnico/
Dr. Sergio Olivar

Secretaria Técnica/
Dra. María Mercedes González

Pro Secretaria General/
Lic. Ana María Turdó

Pro Secretaria Académica/
Lic. Yanina Martínez

Pro Secretario de Ciencia y Tecnología/
Cdor. Adrián Sancci

Pro Secretario de Desarrollo Universitario/
Lic. Juan Boasso

Pro Secretaria Administrativa/
Cdora. Rosana Ibañez

Pro Secretario de Informática y Comunicaciones/
Ing. Claudio D'amico

Pro Secretario de Relaciones Internacionales/
Mag. Federico Scremin

Autoridades DIIT

Decano/
Mag. Gabriel Blanco

Vicedecano/
Mag. Jorge Eterovic

Secretaria Académica/
Mag. Carolina Vicente

Secretaria de Investigaciones/
Dra. Bettina Donadello

Secretaria Administrativa y de Extensión/
Cdora. Mariángeles Vanesa Gallo

Coordinadora Ing. en Informática/
Ing. Andrea Vera

Coordinador Ing. Electrónica/
Ing. Hugo Tantignone

Coordinador Ing. Industrial/
Ing. Mauro Vidal

Coordinador Ing. Civil/
Ing. Fabián Montero

Coordinador Ing. Mecánica/
Ing. Guillermo Rodofile

INGENIERÍA DE REQUISITOS

El camino hacia un software de calidad comienza en un punto de partida confiable

Primera edición

Autora
Gladys N. Kaplan

1 de febrero de 2025

Kaplan, Gladys N.
Ingeniería de requisitos : el camino hacia un software de calidad comienza en un punto de partida confiable / Gladys N. Kaplan. - 1a ed. - San Justo : Universidad Nacional de La Matanza, 2025.
Libro digital, PDF

Archivo Digital: descarga y online
ISBN 978-631-6611-32-1

1. Ingeniería. 2. Ingeniería de Software. I. Título.
CDD 005.10285

Autora
Dra. Gladys N. Kaplan

©Universidad Nacional de La Matanza
Florencio Varela 1903 (B1754JEC)
San Justo/Buenos Aires/Argentina
Telefax:(54-11) 4480-8900
www.unlam.edu.ar

ISBN: 978-631-6611-29-1

PRÓLOGO

El desarrollo de software ha sido, desde sus inicios, una disciplina llena de desafíos. La llamada "crisis del software" hace referencia a los constantes fracasos en proyectos de desarrollo: sobrecostos, incumplimiento de plazos, software de baja calidad y productos que no satisfacen las necesidades del usuario. A pesar de los avances en metodologías, herramientas y enfoques de desarrollo, muchos de estos problemas se han mantenido en el tiempo como un desafío recurrente dentro de la industria.

Las organizaciones que dependen de software de mala calidad enfrentan una serie de problemas que impactan directamente en su productividad, costos y reputación**.** Sistemas inestables, errores frecuentes y funcionalidades mal diseñadas generan interrupciones operativas, pérdida de datos y frustración en los usuarios. Además, el costo de corregir errores aumenta exponencialmente a medida que el software avanza en su ciclo de vida, lo que obliga a destinar recursos a mantenimiento en lugar de innovación. La falta de alineación entre los requisitos del negocio y la implementación técnica también puede llevar a decisiones estratégicas erróneas, dificultando la competitividad en mercados cada vez más exigentes. En un mundo donde la tecnología es el eje de las operaciones, depender de software deficiente no solo es un riesgo, sino una amenaza para la sostenibilidad de cualquier organización.

Este libro invita a reflexionar si esta crisis es un problema inevitable y permanente dentro del desarrollo de software, o si es posible encontrar soluciones efectivas para mitigar sus efectos y mejorar la tasa de éxito de los proyectos.

Exite un gran desafío en cambiar la forma en que abordamos la construccion de software. La definición de requisitos no se trata solo de definir qué quiere el cliente, sino de entender qué necesita realmente.

Aquí es donde entra en juego la Ingeniería de Requisitos. No es una moda, no es un trámite burocrático, y definitivamente no es opcional. Es la base sobre la que se construye cualquier software exitoso. Sin requisitos claros, bien definidos y correctamente gestionados, lo que estamos haciendo no es ingeniería, sino una apuesta al azar con recursos, tiempo y credibilidad en juego.

Más que una simple recopilación de conceptos, este libro pretende generar conciencia sobre el papel crucial de los requisitos en el ciclo de vida del software. En un mundo donde la tecnología avanza a pasos agigantados y las expectativas de los clientes son cada vez más exigentes, la Ingeniería de Requisitos se consolida como una disciplina imprescindible para construir productos software de calidad.

El objetivo de esta obra es ayudar a estudiantes, profesionales y cualquier persona interesada en consolidar sus conocimientos sobre la Ingeniería de Requisitos. A través de un enfoque claro y estructurado, el libro busca transmitir la teoría y también brindar herramientas prácticas que permitan comprender la importancia de los requisitos como un punto de partida confiable para el desarrollo de sistemas de software exitosos.

Mg. Gabriel Blanco
Decano

Mg. Jorge Eterovic
Vicedecano

Ing. Alfredo E. Vázquez
Profesor Emérito

Contenido

Capítulo 1

¿La "crisis del software" se ha convertido en una enfermedad crónica?

Grady Booch afirma que "el desarrollo de software ha sido, es y probablemente será, fundamentalmente difícil". Diferentes autores [Parn72] [Broo95] [McCo96] [Larm03] [Cusu04] han escrito acerca de la historia de la Ingeniería de Software con el objetivo de marcar los avances y retrocesos producidos. Solo a efectos de comprender esta evolución, se analiza la crisis del software como uno de los desencadenantes de estos cambios. En 1968, en la primera conferencia de la OTAN (Organización del Tratado del Atlántico Norte), Friedrich Bauer [Baue69] [Naur69] habló por primera vez del conjunto de dificultades o errores ocurridos en la planificación, estimación de costos, productividad y calidad del software, coincidiendo con lo que luego se conoció como la *crisis del software*. Algunos de los problemas se referían a proyectos de software que sobrepasaban el presupuesto; resultados poco confiables; grandes sistemas de software que resultaban difíciles y caros de mantener; software que no satisfacía las crecientes necesidades del cliente; dependencia del software con el hardware; aumento de la demanda de nuevo software que superaba la capacidad de generarlo; proyectos de software que en promedio

sobrepasaba el 50% de su cronograma original; software, que en muchas ocasiones, nunca se entregaba; entre otros.

La síntesis histórica que sigue comienza en 1970, ya que toma como primer hito la publicación de Royce [Royc70]. Hasta ese momento la construcción de software se realizaba en pequeños ciclos de ***Analysis and Coding*** (análisis y codificación) el cual reemplazó al ciclo ***Code and Fix*** (codificar y corregir) utilizado durante la década 60. Royce propone abrir la construcción de software y define un ciclo abierto con las siguientes fases [1] : Software Requirements, Analysis, Program Design, Coding, Testing and Operations (requisitos del software, análisis, diseño de programas, codificación, pruebas y operaciones). La propuesta es que cada fase se complete antes de pasar a la siguiente. Royce considera que la mejor forma de construir software es sin retroalimentación entre fases, pero reconoció que era "arriesgada e invitaba al fracaso". Por tal motivo, en el mismo artículo describe la manera de iterar entre fases, pero haciendo mención a la incidencia de la retroalimentación en el costo del software indicando que retornar a etapas anteriores puede volver el proceso prácticamente imposible de sostener. La propuesta de Royce se puede observar en la Figura 1, donde, además de abrir las fases del proceso, agrega un ciclo de prueba inicial denominado *Diseño Preliminar del Programa* y lo ubica entre las fases de Requisitos del Software y la de Análisis. El objetivo de este ciclo es controlar el costo de posibles errores u omisiones en las fases siguientes. De esta manera recomienda hacer dos veces el mismo trabajo, la primera vez para comprobar la viabilidad del proceso con un

[1] La primera mención conocida que describe el uso de las fases de la Ingeniería de Software fue realizada por Herbert D. Benington en el "*Symposium on advanced programming methods for digital computers"* en 1956.

prototipo utilizable y la segunda para construirlo. Todo el conocimiento obtenido en esta fase es utilizado para alimentar las posteriores.

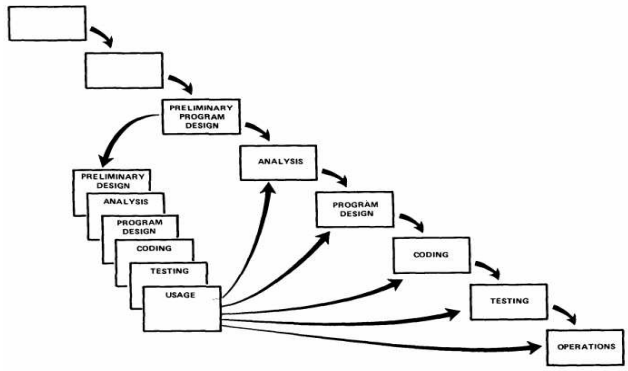


Figura 1 - Modelo propuesto por Royce [Royc70]

Las fases propuestas por Royce fueron conocidas en 1975 como ***Software Develop Life Clicle*** (SDLC) o ciclo de vida del desarrollo del Software, el cual también ha sido fuertemente cuestionado aludiendo que en vez de ayudar perjudica a la construcción del software al hacer muy dificultosa la incorporación de cambios en los requisitos [Glad82]. McCracken y Jackson también argumentaron en contra del modelo de Royce haciendo referencia a una "cascada sofocante" [McCr80] y subrayando la posición subordinada de los modelos iterativos. Brooks dejó muy claro su punto de vista acerca del ***Waterfall Model*** (Modelo en Cascada) en el discurso de apertura en la Conferencia Internacional de Ingeniería de Software de 1995 al decir que "¡El modelo de cascada está mal!". Royce nunca menciona el tan conocido “modelo en cascada” ya que esta denominación surgió con posterioridad cuando se relacionó el trabajo de Royce con una “cascada” de actividades secuenciales [Bell76]. Luego, el modelo en cascada como tal se popularizó a través de la norma estadounidense DoD-STD-2167 en el año 1985 donde se lo utilizó con una sola iteración, generando la duda

si se debía completar el ciclo en una sola iteración o si podían sucederse otras. Este estándar influenciaría a varios otros estándares significativos de la industria como el JSP-188 (Gran Bretaña), V-Model (Alemania), GAM-T-17 (Francia), entre otros. Según Larman y Basilli [Larm03] a pesar de la rigidez del modelo en cascada, se puede ver en el artículo de Royce indicios de desarrollo iterativo, retroalimentación y adaptación y citan un fragmento de la conversación que tuvieron con su hijo Walker Royce:

> *"Él siempre fue un defensor del desarrollo iterativo, incremental, evolutivo. Su artículo describe la cascada como la descripción más simple, pero eso no funcionaría para todos los proyectos, excepto aquellos más sencillos. El resto de su trabajo describe las [prácticas iterativas] en el contexto del modelo de contratación del gobierno de los 60s/70s".*

Las diferentes interpretaciones de la propuesta original de Royce y lo que hoy se conoce como *modelo en cascada* se mantienen en la actualidad, como es el caso de los libros de Ingeniería de Software de Pressman y Sommerville. En Pressman [Pres20] el modelo en cascada prevé ciclos con retroalimentación indirectos, pero indica que “la inmensa mayoría de las organizaciones que aplica este modelo de proceso lo trata como si fuera estrictamente lineal”. En Sommerville [Somm11] las fases del modelo original reflejan las actividades fundamentales del desarrollo y describe que cada fase itera hasta asegurar que está completa, y recién entonces se pasa a la fase siguiente, pero menciona que en la práctica las fases se retroalimentan iterando entre ellas ya que una nutre a la otra.

En 1986 Parnas y Clements publican el artículo "Un proceso de diseño racional: cómo y por qué fingirlo" donde describen que, aunque creen

en el ideal del modelo en cascada (especificaciones completas, correctas y claras antes del desarrollo), es poco práctico. Algunas de sus razones:

- Los usuarios de un sistema rara vez saben exactamente lo que quieren y no pueden expresar todo lo que saben.
- Incluso si pudiéramos establecer todos los requisitos, hay muchos detalles que solo podemos descubrir una vez que están bien implementados.
- Incluso si supiéramos todos estos detalles, como humanos, no podemos dominar tanta complejidad.
- Incluso si pudiéramos dominar toda esta complejidad, las fuerzas externas conducen a cambios en los requisitos, algunos de los cuales pueden invalidar decisiones anteriores. y comentó que, por todas estas razones, "la imagen del diseñador de software que deriva su diseño de una manera racional y libre de errores de una declaración de requisitos es bastante poco realista".

La expresión “incluso si pudiéramos...” expone un conjunto de falencias en los procesos de construcción de software, particularmente en la primera fase del proceso, lo que luego fue absorbido por la Ingeniería de Requisitos. Pero esta necesidad de mejorar y asegurar los procesos de requisitos ya había sido declarada una década antes. En 1976 se llevó a cabo un estudio empírico de casos realizados en 1973 y vueltos a analizar en 1975 [Bell76]. Dividieron los problemas en tempranos y tardíos, arrojando en las revisiones tempranas que aparecía con más fuerza los problemas en los requisitos del tipo: *faltantes*, *incompletos*, *inadecuados* o errores del tipo *poco claros* y, en las revisiones posteriores superó ampliamente la categoría *incorrectos*

seguidos por *omitidos* e *incompletos*. En este estudio determinaron empíricamente que los problemas en los requisitos son una realidad y que requieren una revisión continua. Además, determinaron que los problemas detectados eran muy similares en todos los proyectos. Los requisitos cambiaban durante todo el desarrollo y los errores eran detectados cuando este se acercaba al diseño. Esto mostró la necesidad de contar con técnicas para identificar deficiencias en los requisitos lo más temprano posible para reducir costos y mejorar la calidad de los requisitos.

En 1975, Basili [Basi75] recomendó la ***Iterative improvement techniques*** (Técnica de mejora iterativa) como un medio práctico para realizar un enfoque de refinamiento gradual y top down [2] para el desarrollo de software. De esta manera se pueden *analizar los requisitos para cada iteración*. El mecanismo consiste en crear un esqueleto inicial que sirva de guía para todo el proceso de construcción. Sobre este esqueleto se va iterando hasta tener la versión final. Crea una *lista de control del proyecto* con todas las tareas necesarias para implementar el software deseado, y el proyecto finaliza cuando la lista está vacía. Las tareas pueden ser un rediseño, el diseño e implementación de futuras funcionalidades, la incorporación de funcionalidades que se han perdido o no han sido vistas en la implementación existente, y soluciones a problemas sin resolver. Cada iteración consiste en seleccionar y remover la próxima tarea de la lista, diseñando la implementación de la tarea seleccionada. Según Basili, este esqueleto inicial puede traer algunos problemas, como ser una

[2] Un enfoque de top-down (descendente) comienza con el panorama general y luego se divide en segmentos más pequeños. Un enfoque bottom-up (ascendente) es la unión de sub-sistemas para dar lugar a sistemas más complejos.

mirada parcial del problema en estudio. El esqueleto debe tener los puntos clave del problema y las restricciones. En cada iteración participan los usuarios y se incorporan las modificaciones necesarias. La utilización de la mejora iterativa al desarrollo de software, según los autores, es práctica y eficiente; fomenta la generación de un producto fácilmente modificable ya que, en cada iteración, se realizan modificaciones de diseño junto con la incorporación de nuevas funcionalidades.

Según Larman [Larm03], el ***Iterative Development*** (Desarrollo Iterativo) ya estaba presente en 1968 en un informe de Randell y Zurcher en IBM. Un año más tarde, Lehman retomó este trabajo y en un informe interno para la gerencia insistió en recomendar el Desarrollo Iterativo diciendo que “el enfoque básico reconoce la inutilidad de separar los procesos de diseño, evaluación y documentación en el diseño de sistemas de software. El proceso de diseño está estructurado por un modelo en expansión sembrado por una definición formal del sistema, que proporciona un primer modelo funcional ejecutable. Se prueba y se expande aún más a través de una secuencia de modelos, que desarrollan una cantidad creciente de funciones y una cantidad creciente de detalles sobre cómo se debe ejecutar esa función. Finalmente, el modelo se convierte en el sistema”.

En realidad, el enfoque iterativo surgió del trabajo de Walter Shewhart en la década del 30, padre del Control Estadístico de Procesos (SPC), quien era un experto en calidad de Bell Labs. Este propuso una serie de ciclos cortos para mejorar la calidad: "Plan-Do-Study-Act" (PDSA) [Shew39] como una metodología para la mejora continua. A partir de la década del 40, otro gurú de la calidad, W. Edwards Deming, popularizó y adaptó el ciclo de Shewhart, que a menudo se conoce como el Ciclo

de Deming o PDCA ("Plan-Do-Check-Act") [Demi50], aplicándolo no solo a la manufactura, sino a todos los aspectos de la organización. Otros autores, como Tom Gilb y Richard Zultner, exploraron el enfoque PDCA para el desarrollo de software [Larm03].

Mills, quien trabajó en IBM desde 1964 hasta 1987 y donde alcanzó el estatus de investigador, se desempeñó en importantes cargos. En su vasta experiencia en proyectos de gran envergadura, aplicó el ***Incremental Development*** (Desarrollo Incremental) y la teoría estadística a las pruebas de software. En su conocida "Programación de arriba hacia abajo en grandes sistemas" publicado en 1970 promueve primero el desarrollo iterativo, pero no aclaró que se debía evitar una gran especificación inicial. Tampoco especificó el tamaño de la iteración. Luego, en 1976 pone el énfasis en el Desarrollo Incremental con la participación continua del usuario y el rediseño del software de acuerdo a los costos de cada incremento. En 1987 propone el método de desarrollo de software ***Cleanroom*** (Sala limpia) [Mill87] en uso en los laboratorios de IBM y de la NASA que proporciona una versión incremental de funciones de software o subsistemas (desarrollado a través de refinamiento gradual) para separar los equipos internos de garantía de calidad que aplican medidas estadísticas y análisis como base para certificar sistemas de software de alta calidad. Se basa en método formales[3] para el desarrollo del software con el objetivo de evitar la mayor cantidad de errores posibles durante la construcción del mismo. Tanto Basili como Mills observaron que un problema importante de la construcción de software lineal es que no se valida con el cliente hasta ser puesto en producción. El modelo incremental actúa como un mecanismo de validación parcial del software a medida que se

[3] Los métodos formales permiten representar la especificación del software, verificación y diseño de componentes mediante notaciones matemáticas.

construye y permite detectar tempranamente errores en los requisitos. Por otro lado, Glass [Glas69] menciona que el desarrollo incremental vale la pena, obliga a una exhaustiva revisión del sistema y evita el desaliento que produce la implementación y la administración. Desarrollar con un enfoque incremental permite al desarrollador aprovechar lo que se aprendió durante el desarrollo de las versiones anteriores del sistema (ver Figura 2).

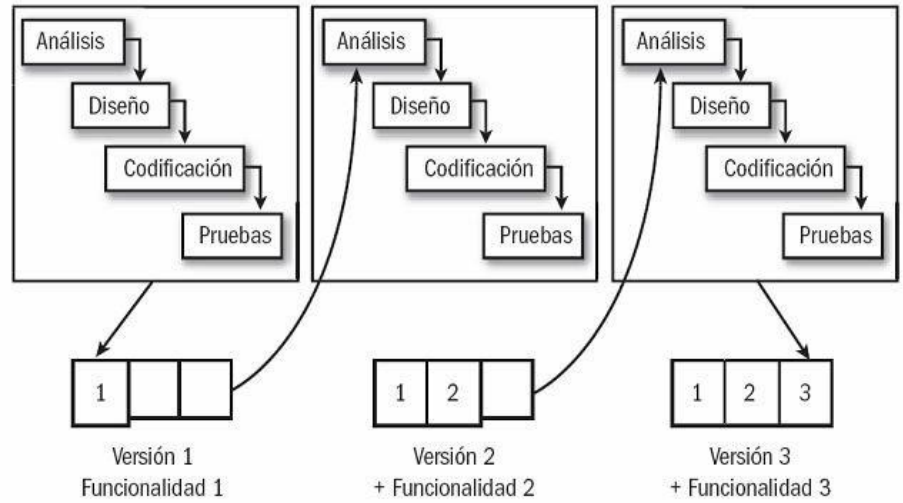


Figura 2 - Modelo Incremental

El aprendizaje proviene tanto del desarrollo como del uso del sistema, siempre que sea posible. Los pasos clave en el proceso fueron: a) comenzar con una implementación simple de un subconjunto de los requisitos de software y b) cada subconjunto genera un entregable que va evolucionando hasta implementar el sistema de software completo. En cada incremento, se realizan modificaciones de diseño junto con la adición de nuevas capacidades funcionales.

La primera publicación acerca del ***Iterative Incremental Development*** (Desarrollo Iterativo Incremental) (IID) fue en 1978, en la columna del UK’s Computer Weekly donde escribía Tom Gilb. Según Larman y Basili [Larm03] en la década del 50 la construcción del avión

hipersónico X-15 utilizó un enfoque IID. Este proyecto, por su éxito, se convirtió en un hito de la década y aunque no era de software en 1960 algunos miembros del personal del proyecto Mercury de la NASA retomaron la experiencia. Los resultados en cuanto a costos y satisfacción del cliente lo popularizaron empresas como IBM y TRW. El IID se convirtió en la mejor manera de construir software.

En 1982, Swartout y Balzer argumentaron que la especificación y el diseño tenían una interacción necesaria, y promovieron un enfoque iterativo y evolutivo a la construcción de los requisitos. Esta metodología de desarrollo se la denominó ***Prototyping*** (Prototipado) [Goma81] [Floy84] y permite construir el sistema de software de una manera rápida y a un bajo costo. El mismo software actúa como un elemento de comunicación entre el desarrollador y el cliente, permitiendo validar, de manera constante, los requisitos incorporados. Comienza con pocos requisitos, los más conocidos o esenciales, y los va completando de una forma evolutiva. De esta manera se asegura la comprensión de la relación entre las funciones del software y las tareas existentes.

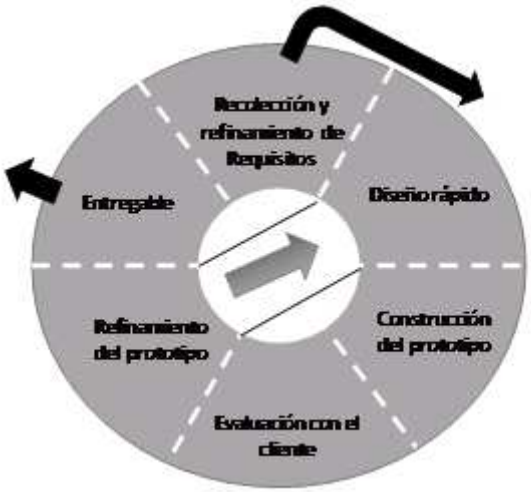


Figura 3 - Modelo Prototipado Evolutivo

El ***Operational Specifications for Rapid Prototyping*** (Especificaciones operativas para la creación rápida de prototipos) [Balz83] [Zave84] (ver Figura 3) supone la existencia de un lenguaje de especificación formal para soportar el desarrollo evolutivo de especificaciones en un prototipo de implementación. Las especificaciones en el lenguaje de especificación están codificadas, y cuando se evalúa computacionalmente, constituye un prototipo funcional del sistema de software. Cuando tales especificaciones se pueden desarrollar y procesar de forma incremental, el sistema de software resultante puede ser refinado y evolucionado en sistemas de software funcionalmente más completos. Sin embargo, los sistemas de software emergentes siempre están operativos de alguna forma durante su desarrollo. Las variaciones dentro de este enfoque representan esfuerzos donde el prototipo es el fin buscado, o donde los prototipos especificados se mantienen operativos pero refinados en un sistema de software completo.

En 1988 se publica el ***Spiral Model*** (Modelo en Espiral) [Boeh88] (ver Figura 4) basado en la experiencia de 14 años en el uso del modelo en cascada. Boehm se basa en el modelo de Royce, pero incorpora una fase de análisis de riesgo para evaluar los objetivos y restricciones definidas durante el planeamiento. Esto permite identificar áreas de incertidumbre que son potenciales riesgos para el proyecto. Luego, se debe evaluar el costo del riesgo construyendo un prototipo, realizando una simulación, o cualquier otra técnica de resolución de riesgo. Si el rendimiento o el riesgo de interfaz de usuario dominan fuertemente el desarrollo del programa o el riesgo de control interno de la interfaz, el siguiente paso puede ser un desarrollo evolutivo: un esfuerzo mínimo para especificar la naturaleza general de un producto, un plan para el

siguiente nivel de prototipado y el desarrollo del prototipo más detallado para continuar resolviendo el principal problema de riesgo.

El enfoque en general es por ciclos de desarrollo iterativos como una espiral en expansión, con ciclos internos que denotan análisis temprano del sistema de software y creación de prototipos, y ciclos externos que denotan el ciclo de vida clásico del software. La dimensión radial indica los costos de desarrollo acumulativos, y la dimensión angular indica progresos realizados en la realización de cada espiral de desarrollo. Boehm utiliza la propuesta de Royce en el cuadrante de desarrollo donde aplica un proceso secuencial, pero además incorpora la estrategia incremental al permitir realizar el espiral tantas veces como sea necesario, de manera incremental, hasta construir el software deseado.

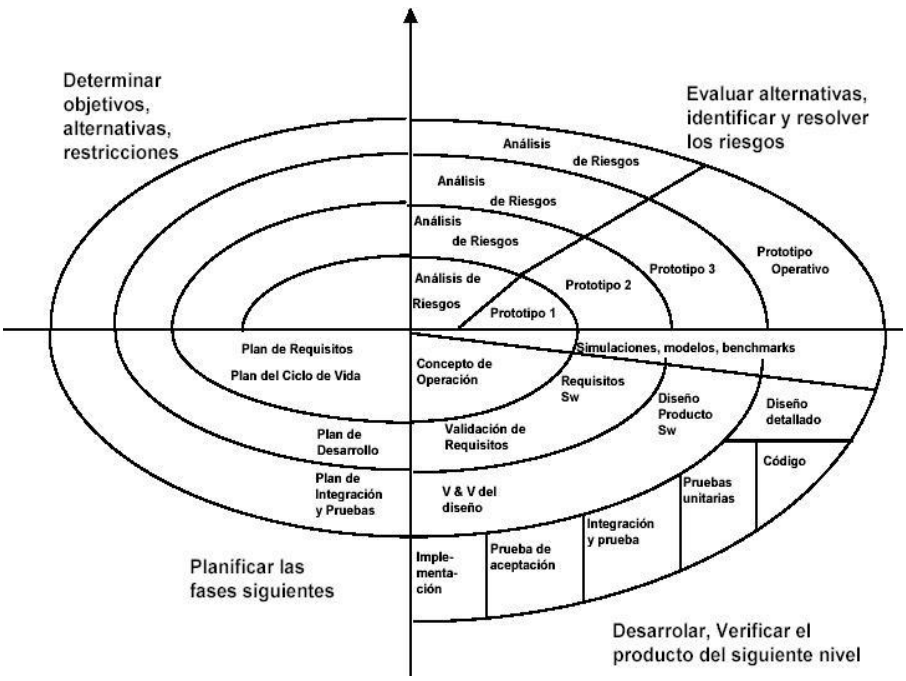


Figura 4 - Modelo en Espiral [Boeh88]

La filosofía conocida como “compre y no construya" divulgada por Brooks en 1987, fue aplicada en los 90 en el ***Component-based Development*** (Desarrollo basado en Componentes) (CBD) [Brow97] [Atki00]. Se define como un conjunto de tecnologías, herramientas y técnicas que permiten a las organizaciones de desarrollo pasar por el proceso de creación, catalogación, ensamblaje e integración de componentes de software. Propone una integración de componentes existentes para satisfacer los requisitos del software de una manera más segura y rápida, reduciendo el trabajo de construcción aproximadamente en un 70%. En la década del 80 e inicios de los 90 existía una visión acerca de que la forma más adecuada de construir software era bajo un plan de proyecto riguroso, aseguramiento de la calidad, procesos de desarrollos rígidos y muy controlados. Esta idea se genera debido a que las grandes difusiones de la evolución en la construcción del software estuvieron centradas en la construcción de grandes sistemas de software de larga duración [Somm11]. La aplicación de esta forma de trabajar en sistemas de software pequeños y medianos se tornó difícil y costosa, donde hacer los cambios requeridos por un entorno dinámico y la presión del cliente en ir teniendo resultados, se volvió muy dificultoso.

En 2001 miembros de la comunidad adoptaron el nombre de métodos ágiles [Beck01] y poco después formaron la alianza ágil constituida por Extreme Programming (XP), Scrum, Dynamic Systems Development Method (DSDM), Adaptive Software Development (ASD), Crystal, Feature-Driven Development (FDD), Pragmatic Programming, entre otros, para desarrollar el Manifiesto Ágil. Otros métodos agiles conocidos son Agile Unified Process, Crystal Clear, Lean Software Development (LSD), Kanban, Open Unified Process (OpenUP),

Método de desarrollo de sistemas dinámicos (D SDM), G300, 6D-BUM, PMI Agile, etc. Estas metodologías se basaron en el desarrollo iterativo e incremental, la cual se centra en las personas que participan en el desarrollo más que en los procesos y herramientas. La propuesta de esta metodología se focalizó en:

- Individuos e interacciones sobre procesos y herramientas.
- Entregas constantes sobre documentación.
- Colaboración del cliente sobre negociación.
- Respuesta al cambio sobre seguir un plan.

Estas metodologías ágiles satisfacen las necesidades del cliente mediante entregas continuas y la adaptación al cambio, pero este proceso está inmerso en la producción del software manteniendo la mirada en el producto a construir. Para Tom Gilb [Gilb81] “Hay algo extraño en una cultura de software que está tan obsesionada con la novedad y el determinismo tecnológico que perpetuamente pierde el conocimiento del pasado en una búsqueda interminable de la reinvención de la rueda. La carrera a partir del año 2000 para ‘agilizarnos’ y distanciarnos lo más posible de cualquier vieja perspectiva de la Ingeniería de Software, siempre con el fantasma del modelo en cascada, ha resultado en la negligencia de una gran cantidad de conocimiento y sabiduría de los grandes ingenieros/as de sistemas de las décadas del 60 al 80. Es nuestra pérdida. Estamos olvidando cosas que son mejores de lo que creemos saber ahora”. Esta opinión se sustenta en que Gilb es uno de los primeros y más activos promotores del IID. Su material fue probablemente el primero con un claro sabor de iteración ágil, ligera y adaptativa con resultados rápidos, similar al de los métodos IID más nuevos [RUP98] [Jaco99]. Aunque algunos prefieren reservar la frase "desarrollo iterativo" simplemente para

reelaborar, en los métodos ágiles modernos el término implica no solo revisar el trabajo, sino también el avance evolutivo, un uso que data de 1968. IID representa mejoras y ampliaciones de funcionalidad en cada ciclo. Parece ser que el modelo IID fue el que revolucionó la historia de la construcción del software. Se puede observar como impactó en la mayoría de las propuestas, inclusive hasta nuestros días. La estrategia Iterativa asegura el proceso, hacer las cosas hasta lograr la calidad deseada. El incremental asegura los requisitos del software al ir avanzando de a poco y revisando y adaptando los requisitos comprometidos.

En los últimos años (2010 a 2024) los cambios producidos se han corrido de eje. Ya no están tan centrados en los procesos, sino más bien en las personas, en la conformación de equipos de trabajo, la autogestión, la colaboración, la confianza. Este es el caso de Spotify, Amazon, Google y Microsoft que ganaron en Wall Street el primer lugar al utilizar las prácticas de desarrollo ágil en toda la organización. Entre las metodologías para grandes desarrollos están: Scrum, SAFe (Scaled Agile Framework), Kanban, Less (Large Scale Scrum), Disciplined Agile (DA), Scaled Agile Framework, Spotify Model.

En la Tabla 1 se puede observar la evolución de la construcción del software acotada a algunos de los principales hitos. Cabe aclarar que las fechas, en algunos casos, son aproximadas o tienen que ver con su popularización. En esta tabla se encierran muchos aspectos relevantes relacionados con la construcción de software y donde los/las líderes del pensamiento de la disciplina han sembrado los fundamentos para construir software de *calidad* al *menor costo* posible. Una ecuación que debe estar siempre presente.

Cerrando esta síntesis, se retoma a la pregunta inicial *¿La "crisis del software" se ha convertido en una enfermedad crónica?* Para responder no alcanza con estadísticas aisladas ni datos globales, se requiere evaluar múltiples variables que afectan a la construcción del software y en muchos casos, en simultáneo. Analizar algunas de estas variables es el objetivo del presente libro.

Década	Algunos hitos que acompañaron	Mercado
1950	Tarjetas perforadas. Altos costos.	1957 se utilizó el Incremental en IBM.
1960	Codificar y corregir. Ingeniería de Software. …	1968 se utilizó desarrollo Iterativo en IBM.
1970	Sistemas multiusuarios. Sistemas en tiempo real. Sistemas de Base de datos. Internet. Programación estructurada. Primeras menciones de la Ingeniería de Requisitos. …	1970 se utilizó desarrollo Iterativo en IBM. 1976 se utilizó IID en IBM.
1980	Sistemas Distribuidos. Bajan los costos del hardware. Calidad del software. Sistemas expertos. …	
1990	Redes neuronales. Entorno cliente servidor. WWW. Orientación a Objetos. Rational Unified Process (RUP). Ingeniería de Requisitos. …	
2000	Sistemas potentes. Reutilización. Metodologías Ágiles. Automatización y DevOps. …	

2010	Omnipresencia de la web. Desarrollo en la nube. Microservicios y Arquitectura Basada en Contenedores. Arquitectura orientada a servicios (SOA). Low-Code y No-Code Development. IA para el Procesamiento del Lenguaje Natural (NLP), aprendizaje automático, automatización de pruebas, DevOps. …	2011 Ágiles a gran escala: Microsoft, Apple, Spotify, Google, Facebook y Amazon.
2020	Ingeniería de Software Ecológico. Computación Cuántica. Plataformas de Inteligencia de Ingeniería de Software. Adopción masiva de inteligencia artificial. Programación Conversacional. Observabilidad en Tiempo Real. Edge Computing. …	

Tabla 1 - Hitos de la Ingeniería de Software

Referencias

[Atki00] Atkinson C., Bayer J., Muthig D.(2000). “Component-based product line development: The KobrA approach”, Springer.

[Balz83] Balzer, R., Cheatham, T.E., Green, C. (1983). “Software technology in the 1990s: using a new paradigm”, IEEE Computer.

[Basi75] Basili V. and Turner J. (1975). “Iterative Enhancement: A Practical Technique for Software Development”, IEEE Trans. Software Eng., pp. 390- 396.

[Baue69] Bauer F. L., Bolliet L., Helms H. J. (1969). “Software Engineering”. Report on a conference sponsored by the NATO SCIENCE COMMITTEE Garmisch, Germany, 7th to 11th October 1968. Editors: Peter Naur and Brian Randell

[Bell76] Bell, T.E., Thayer, T. A. (1976). “Software Requirements: are they really a problem?”, Second International Conference on Software Engineering.

[Beck01] Beck, K. et al. (2001). “Manifesto for agile software development”. Agile Alliance. https://agilemanifesto.org

[Boeh88] Boehm, B.W. (1988). "A Spiral Model of Software Development and Enhancement", IEEE Computer, Vol.21, Nº5, pp.61-72.

[Broo95] Brooks, F. P. (1995). "The Mythical Man-Month: Essays on Software Engineering". (20th Anniversary Edition). Addison-Wesley.

[Brow97] Brown A.W.; Short K. (1997). "On components and objects: the foundations of component-based development", Proceedings Fifth International Symposium on Assessment of Software Tools and Technologies, DOI: 10.1109/AST.1997.599921.

[Cusu04] Cusumano, M. A. (2004). "The Business of Software: What Every Manager, Programmer, and Entrepreneur Must Know to Thrive and Survive in Good Times and Bad". Free Press.

[Demi50] Deming, W.E. (1950). "Elementary Principles of the Statistical Control of Quality", JUSE.

[Floy84] Floyd CA. (1984). "Systematic Look at Prototyping". In: Budde R., Kuhlenkamp K., Mathiassen L., Züllighoven H. (eds) Approaches to Prototyping. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-69796-8_1,1984

[Gilb81] Gilb T. (1981) "Evolutionary Development," ACM Software Eng. Notes, p. 17

[Glad82] Gladden, G.R. (1982). "Stop the life cicle, I want to get off", Software Engineering Note, Vol 7, Nro 2, pp 35 ACM SIGSOFT.

[Glas69] Glass R. (1969). "Elementary Level Discussion of Compiler/Interpreter Writing," ACM Computing Surveys, pp. 64-68.

[Goma81] Gomaa, H., Scott, D.B. (1981). "Prototyping as a Tool in the Specification of User Requirements", Fifth International Conference on Software Engineering, IEEE Computer Society Press, pp.333-342.

[Jaco99] Jacobson, I., Booch, G., Rumbaugh, J. (1999). "The Unified Software Development Process", Addison-Wesley, Reading, MA, 1º edición.

[Larm03] Craig Larman, Victor R. Basili (2003). "Iterative and Incremental Development: A Brief History", pp. 47-56, vol. 36, DOI Bookmark: 10.1109 / MC.2003.1204375, IEEE.

[McCr80] McCracken D. y Jackson M. (1980). “A minority dissenting position” in “Systems analysis and design” William W. Cotterman, Institute for Certification of Computer Professionals, IEEE Computer Society, Association for Computing Machinery.

[McCo96] McConnell, S. (1996). “Rapid Development: Taming Wild Software Schedules”. Microsoft Press.

[Mill87] Mills, Harlan D.; Dyer, M.; and Linger, R. C. (1987). "Cleanroom Software Engineering”. The Harlan D. Mills Collection.

[Naur69] Naur, P., & Randell, B. (Eds.). (1969). “Software Engineering: Report on a Conference Sponsored by the NATO Science Committee”. Garmisch, Germany.

[Parn72] Parnas, D. L. (1972). “On the Criteria to Be Used in Decomposing Systems into Modules”. Communications of the ACM, 15(12), 1053–1058.

[Pres20] Pressman, R. S., & Maxim, B. R. (2020). “Software engineering: A practitioner’s approach” (9th ed.). McGraw-Hill Education.

[Royc70] Royce, W.W. (1970) “Managing the Development of Large Software Systems: concepts and techniques”, IEEE WESCON, Los Angeles, CA.

[RUP98] Rational Software. (1998). “RUP: Rational Unified Process”, IBM Corporation, http://www-306.ibm.com/software/rational.

[Shew39] Shewhart, W. A. (1939). “Statistical Method from the Viewpoint of Quality Control”, Department of Agriculture. Dover.

[Somm11] Sommerville, I. (2011). “Ingeniería de Software”, Pearson, Edition 9.

[Zave84] Zave, P. (1984). “The Operational Versus the Conventional Approach to Software Development”, Communications of the ACM, 27, 104-118.

Capítulo 2

Incidencia de los requisitos en los proyectos

A pesar de los innumerables esfuerzos por mejorar la calidad de los sistemas de software, la tasa de fracaso continúa siendo alta. Antes de analizar algunas estadísticas, es importante dejar claro que significa que un proyecto sea exitoso, desafiante o fallido.

Fuente	Proyecto Exitoso	Proyecto Desafiante o con problemas	Proyecto Fallido o Cancelado
Standish Group	Cumple alcance, tiempo y presupuesto.	Excede tiempo o costo, o no cumple completamente el alcance.	Cancelado, no se completa o no se usa tras su entrega.
PMI	Cumple tiempo, costo, alcance y satisface a los stakeholders.	Problemas en uno o más factores (tiempo, costo, alcance), pero logra objetivos mínimos.	Incumple objetivos principales, o el producto no es útil ni aceptado.
McKinsey & Co.	Entregado con valor comercial y dentro de los límites del presupuesto y cronograma.	Ajustes significativos a los recursos o alcance para completarlo; impacto comercial reducido.	No genera valor, o sus retrasos y sobrecostos lo vuelven irrelevante o perjudicial para la empresa.

Tabla 2 – Definición de proyecto exitoso, desafiante y fallido

En la Tabla 2 se pueden observar pequeñas diferencias en las definiciones de acuerdo a la fuente, pero todas se concentran principalmente en tres factores: alcance, tiempo y costo. La relación entre estos factores determina una figura conocida como *Triángulo de*

Oro[4]. Este triángulo es una herramienta fundamental para comprender los desequilibrios críticos que puede sufrir un proyecto software que lo lleve a fracasar o, al menos, convertirse en uno desafiante.

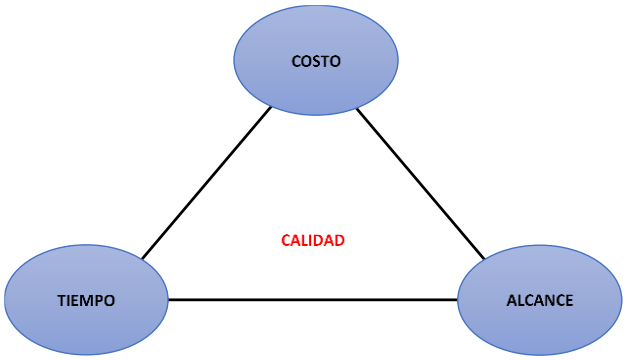


Figura 5 – Triángulo de oro de un proyecto

Cada factor representa:

- *Alcance.* Se refiere a los entregables y tareas que deben completarse para alcanzar los objetivos del proyecto. El alcance puede cambiar si las partes interesadas deciden que quieren ajustar el producto o añadir otro.
- *Tiempo.* Es lo que se tarda en completar las tareas de un proyecto y el proyecto en sí. Esta limitación también se denomina calendario. Una ampliación del alcance puede aumentar los plazos.
- *Costo.* Es la cantidad total de dinero necesaria para completarlo. También se denomina presupuesto. Los costos pueden incluir los salarios de los empleados y el dinero para equipos, herramientas, espacio de oficina y otros recursos. Añadir nuevos miembros a

[4] También conocido como Triángulo de hierro.

un equipo o aumentar el tiempo necesario para completar un proyecto puede repercutir en el costo.

En la Figura 5 se puede observar como estos tres factores están fuertemente interrelacionados y en equilibrio, lo que quiere significa que cuando uno de ellos falla afecta inevitablemente a los otros. El resultado de los proyectos software tiene una historia larga y bien documentada, reflejando cómo ha evolucionado la industria del desarrollo de software desde sus inicios hasta nuestros días. Algunos datos del pasado reciente para analizar [Saxe16]:

- Chaos Report (1994), el 16.2% de los proyectos de software son exitosos, el 52.7% han terminado con problemas y el 31.1% no terminaron nunca.
- Calogero (2000), el 10% terminaron según lo planificado, el 55% con exceso de costo y tiempo y un 35% fueron cancelados.
- Boston Consulting Group (2000), el 67% de los proyectos se consideran fracasados.
- Conference Board Survey (2001), el 40% de los proyectos no cumplen las expectativas del negocio luego de un año en operación.
- Robbins-Gioia Survey (2001), el 51% de las implementaciones de ERP son fallidas.

Información más actual proveniente de diferentes orígenes nos muestra que existen múltiples problemas por el cual puede fracasar un proyecto:

- Forbes (2020), más del 70% de los proyectos de transformación digital fracasan o no alcanzan sus objetivos

debido a problemas en la gestión del cambio y falta de adopción por parte de los usuarios.

- National Audit Office (Reino Unido) (2021), el 57% de los proyectos de software gubernamentales exceden el presupuesto o no cumplen sus objetivos debido a una gestión deficiente.
- Capgemini (2021), 40% de los proyectos tecnológicos-financieros (fintech) no logran alcanzar sus objetivos debido a problemas de integración con sistemas heredados.
- HIMSS (Healthcare Information and Management Systems Society) (2021), 50% de los proyectos de software de salud fallan debido a una gestión inadecuada de los requisitos y regulaciones.
- Standish Group Report (2020), el 31% de los proyectos son exitosos, el 50% tiene problemas y el 19% ha fracasado.
- PMI (Project Management Institute) (2020) y el Standish Group (2021), coinciden en que el 35-40% de los fracasos es por una mala gestión de requisitos, el 30% por falta de comunicación, el 29% de los proyectos fallidos se atribuyen a la falta de compromiso del cliente o del usuario.

En síntesis, el éxito o fracaso de los proyectos software depende fundamentalmente de cómo se relaciona la satisfacción del cliente con los factores del triángulo de oro. En este contexto, una mala definición de requisitos o gestión de los mismos pasa a ser uno de los principales factores de fracaso, ya que, aproximadamente, uno de cada tres fracasos está asociado directamente con ellos.

2.1 ¿Por qué fracasan los proyectos?

Como se ha mencionado, los proyectos de construcción de software fracasan por diversas razones, que pueden abarcar problemas técnicos, organizativos y humanos. Si a estas variables se les agrega una mala estrategia de mitigación de riesgos la posibilidad de alcanzar un proyecto exitoso se torna prácticamente imposible. Según el Project Management Institute (PMI) algunos motivos de fracasos están relacionados con una deficiente gestión del proyecto, falta de planificación, cronogramas irreales o no actualizados, cambio en el alcance del proyecto (scope creep), pobre participación de los usuarios, subestimación de tiempo y recursos, etc.

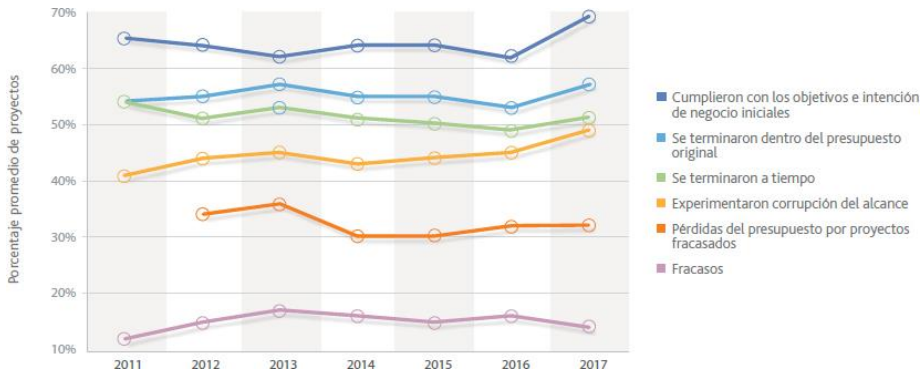


Figura 6 - Métricas de desempeño de Proyectos[4]

En la Figura 6 se puede observar una métrica de desempeño de proyectos realizada desde 2006 a 2017 para analizar el desenvolvimiento de los mismos. Si bien los problemas relacionados con la gestión de proyecto afectan seriamente los resultados esperados,

existen otros problemas que también deben ser cuidadosamente considerados:

- *Comunicación ineficiente*. Malos entendidos, información omitida o que entra en conflicto. Aumenta el tiempo y el costo del software por desgaste y re-trabajo constante.
- *Decisiones equivocadas*. Incorrecta elección de la metodología de construcción, inclusión de módulos inapropiados en el software, etc. Tiempos y recursos mal utilizados y un alto riesgo de no poder terminar el software.
- *Inexperiencia del equipo de desarrollo*. Equipo de desarrollo no tiene las competencias necesarias o experiencia en las tecnologías o metodologías elegidas y el software puede no cumplir con las expectativas de calidad y rendimiento.
- *Poca conciencia acerca de la ética profesional.* No se respeta la ética profesional de la Ingeniería de Software durante la construcción del software.
- *Los nuevos métodos*. Cuando un método o metodología ya está lo suficientemente probada y es segura para ser utilizada, las necesidades reales de las organizaciones cambian.
- *Se deben incluir todas las variables*. Las variables que pueden afectar la construcción del software, como ser las variables directas (interés y disponibilidad de los usuarios, ausencia de procedimientos organizacionales bien definidos, etc.) y las variables indirectas (falta de políticas organizacionales claras, presiones de la competencia, etc.) deben ser tomadas en cuenta a la hora de tomar decisiones.

Se puede observar la gran disparidad de temas que pueden influir en el fracaso de un proyecto software, pero existe una fuerte coincidencia entre los autores que analizan este tema en que el principal factor que influye en el fracaso de los proyectos software radica fundamentalmente en la baja calidad de los requisitos [Boeh81] [GAO92] [Faul92] [Lutz93].

2.2 Costo de corrección de errores en los requisitos

Cuando se hace referencia a errores en los requisitos, habitualmente se desliza la idea de que el/la profesional de software o el proceso utilizado han sido el origen de esos errores. Esta suposición probablemente sea parcialmente falsa ya que muchos de los errores, discrepancias y omisiones (DEO) que aparecen en los requisitos y se propagan hasta el sistema de software final, ya existían en la fuente de información. Pero no es una postura adecuada dejarle la responsabilidad a la fuente de información, sino que en realidad está compartida con los procesos de requisitos que tienen la obligación de ayudar al equipo de requisitos a detectar la mayor cantidad posible de problemas existentes en el contexto, principalmente las discrepancias y omisiones.

Como se muestra en laFigura 7, cuanto más tarde es la detección de un error, más alto es el costo de corrección, con un impacto negativo dentro de la organización que puede afectar la confianza y la cooperación para completar exitosamente el proyecto. La cantidad de errores relacionados con los requisitos del software es muy

significativa, y muchos de ellos se detectan tardíamente cuando podrían haber sido identificados mucho antes [Davi93].

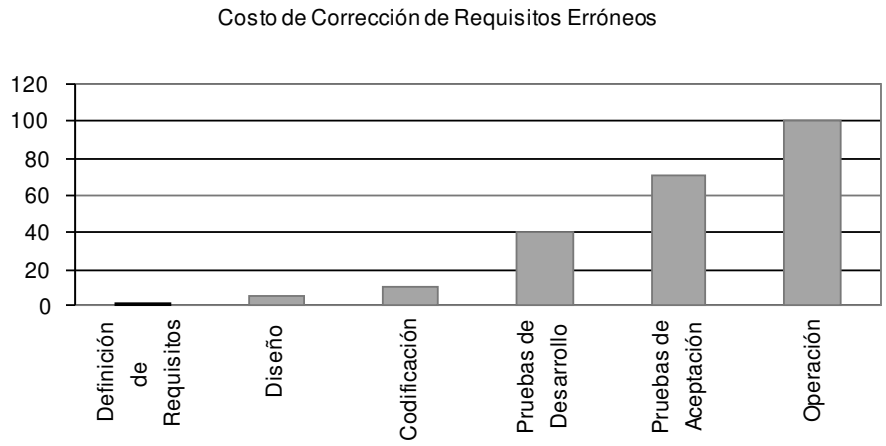


Figura 7 - Costo de corrección de los requisitos [Boeh81]

La incidencia del costo de corrección de los requisitos del software ha sido ampliamente estudiada por diferentes autores. A continuación, algunas reflexiones relacionadas con este tema:

> **Jackson** (1975), "*Algunos proyectos han fracasado porque sus requisitos son inadecuadamente explorados y definidos".*
>
> **DeMarco** (1978), *"El 56% de los errores tienen su origen en la etapa de requisitos*".
>
> **Mizuno** (1983), "*Cuanto más tarde en el ciclo de vida se detecta un error, más costoso es repararlo*: *catarata de errores*".

> **Wieger & Beatty** (2013), "*Los errores de requisitos no detectados a tiempo suelen ser las principales fuentes de costos imprevistos en proyectos de software, destacando la importancia de la validación temprana y continua".*
>
> **Nasa** (2015), *"Los errores en los requisitos identificados durante la fase de pruebas aumentan los costos de corrección en un 30% o más del presupuesto inicial del proyecto".*
>
> **Jones** (2017), "*El costo de corregir un error de requisitos durante la fase de implementación es, en promedio, 50 veces mayor que corregirlo en la etapa de análisis inicial".*
>
> **PMI** (2018), *"El mal manejo de los requisitos es responsable de hasta un 47% de los costos no previstos en los proyectos de software".*
>
> **McConnell** (2019), "*El mal manejo de los requisitos es la causa principal de los defectos críticos en software, lo que puede incrementar exponencialmente los costos si los errores se encuentran tarde en el ciclo de vida del desarrollo*".

En respuesta a la baja calidad de los productos software con altos costos de corrección y mantenimiento, muchos desarrolladores/as de software e investigadores/as han centrado su estudio particularmente en la fase de Ingeniería de Requisitos y en la manera de prevenir que los defectos en los requisitos lleguen a etapas posteriores. De esta manera, se busca garantizar la producción de software de alta calidad y bajo costo basándose en el establecimiento de un punto de partida de alta confiabilidad.

2.3 Factores que contribuyen al éxito de un proyecto

En primera instancia, se podría decir que para que un proyecto sea exitoso se deben evitar todos los factores de fracaso descriptos con anterioridad. El PMI no solo resume lo antes dicho, sino que incorpora algunos aspectos nuevos que son determinantes para el éxito de un proyecto:

- *Definición clara de objetivos y requisitos*. El alcance bien definido y los requisitos detallados ayudan a todos los miembros del proyecto a comprender lo que se espera y a evitar malentendidos.
- *Planificación y gestión adecuada del proyecto*. Un plan sólido con hitos claros, un cronograma realista y una buena asignación de recursos es esencial para mantener el proyecto en curso.
- *Compromiso y apoyo de la alta dirección*. El acompañamiento de los ejecutivos/as de la organización asegura que el proyecto reciba el apoyo necesario en términos de recursos, decisiones y alineación estratégica.
- *Comunicación efectiva.* Mantener a todas las partes interesadas informadas y establecer canales claros de comunicación ayuda a prevenir problemas y permite resolver inconvenientes rápidamente.
- *Equipos con las habilidades adecuadas.* Contar con un equipo con la experiencia técnica y de gestión necesaria garantiza que el desarrollo avance sin contratiempos.
- *Bajar la resistencia al cambio.* Involucrar a los usuarios finales desde el inicio y prepararlos para la transición ayuda a mitigar

la resistencia al cambio y facilita la adopción de las nuevas tecnologías.

- *Monitoreo y control constante*. Hacer un seguimiento constante del progreso, permite detectar problemas a tiempo y corregir desvíos del plan original.
- *Gestión de riesgos.* Identificar, evaluar y mitigar riesgos de manera proactiva evita problemas inesperados.
- *Adaptabilidad y flexibilidad.* Poder adaptarse a cambios en los requisitos o en el entorno es esencial, ya que los proyectos de TI a menudo enfrentan nuevos desafíos o descubrimientos técnicos.
- *Calidad en la entrega.* Asegurarse de que los productos finales cumplan con los estándares de calidad esperados y que funcionen correctamente, así como realizar pruebas exhaustivas, asegura la satisfacción de los usuarios y el éxito del proyecto.
- *Adherencia a las restricciones del proyecto.* Un proyecto exitoso debe finalizar dentro del tiempo planificado, respetando el presupuesto y cumpliendo con el alcance y los estándares de calidad requeridos.
- *Satisfacción del cliente o partes interesadas.* No solo es importante entregar el proyecto a tiempo y dentro del presupuesto, sino que las partes interesadas (cliente, usuarios, etc.) deben estar satisfechas con los resultados del proyecto.

Es importante volver a remarcar la importancia de los requisitos para el éxito de un proyecto. Esto se debe a que actúan como un **pilar fundamental** durante todo el ciclo de vida del software, siendo de vital importancia para garantizar el equilibrio del triángulo de oro del proyecto.

Referencias

[Boeh81] Boehm, B. (1981). "Software Engineering Economics". Englewood Cliffs, NJ: Prentice-Hall.

[Capg21] Capgemini. (2021). "World FinTech Report 2021". Capgemini Research Institute.

[Davi93] Davis, A. M. (1993). "Software Requirements: Objects, Functions and States", Prentice Hall, Englewood Cliffs, NJ, 2º edición.

[DeMa78] DeMarco, T. (1978). "Structured Analysis and System Specification" Yourdon. ISBN 978-0-917072-07-9.

[Faul92] Faulk, S. et al. (1992). "The Core Method for Real-Time Requirements", IEEE Software, Vol.9, Nº5, pp.22-33.

[Forb20] Forbes. (2020). "Why Digital Transformations Fail: Lessons from the Field". Forbes Insights.

[GAO92] US General Accounting Office. (1992). "Mission Critical Systems: Defense Attempting to Address Major Software Challenges", GAO/IMTEC-93-13.

[HIMS20] HIMSS. (2020). "2020 HIMSS Cybersecurity Survey". Healthcare Information and Management Systems Society.

[Jack95] Jackson, M. (1995) "Software Requirements & Specifications. A lexicon of practice, principles and prejudices", Addison-Wesley, Reading, MA/ACM Press.

[Jone17] Jones, C. (2017). "The Economics of Software Quality". Addison-Wesley.

[Lutz93] Lutz, R. (1993). "Analyzing Software Requirements Errors in Safety-Critical Embedded Systems", IEEE First International Symposium on Requirements Engineering, RE'93, IEEE Computer Society Press, Los Alamitos, CA, pp.126-133.

[McCo19] McConnell, S. (2019). "Code Complete: A Practical Handbook of Software Construction". (2nd ed.). Microsoft Press.

[McKi20] McKinsey & Company. (2020). "The Risky Business of Large IT Projects". McKinsey Insights.

[Mizu83] Mizuno, Y. (1983). “Software Quality Improvement”, Computer, Vol. 16, 3.

[NASA15] NASA. (2015). “NASA Systems Engineering Handbook”. NASA-SP-2016-6105.

[PMI18] Project Management Institute. (2018). “Success Rates Rise: Transforming the High Cost of Low Performance”. PMI.

[PMI21] Project Management Institute. (2021). “Pulse of the Profession 2021: Beyond Agility”. PMI.

[Saxe16] Saxena, Deepak & Dempsey, Brian & Mcdonagh, Joe. (2016). “Beyond the One-dimensional Construct of Failure: The Curious Case of Enterprise System Failure Rates”.

[Stan20] Standish Group. (2020). “CHAOS Report: Beyond Infinity”. Standish Group International.

[Wieg13] Wiegers, K. E., Beatty, J. (2013). Software requirements (3rd ed.). Microsoft Press.

Capítulo 3 La Ingeniería de Requisitos en el marco de la Ingeniería de Software

3.1 Proceso ingenieril

Ambas disciplinas, la Ingeniería de Software y como parte de esta, la Ingeniería de Requisitos, se enmarcan en un proceso ingenieril, ya que siguen principios y metodologías similares a otras ingenierías, como la civil o la mecánica. Billy Vaughn Koen, profesor de la Universidad de Texas, define que "el método de la ingeniería es la estrategia para causar, con los recursos disponibles, el mejor cambio posible en una situación incierta o pobremente estudiada" y afirma que el método ingenieril consiste en el uso de heurismos[5] para producir el mejor cambio, con los recursos disponibles, en una situación deficientemente entendida. Los métodos ingenieriles utilizan procesos estructurados, sistemáticos y disciplinados, donde el enfoque estructurado ayuda a minimizar errores, reducir costos y garantizar que el producto final sea fiable y cumpla con los objetivos planteados, mientras que el enfoque sistemático y disciplinado permite construir soluciones eficaces y de

[5] Término asociado a "Heurística". Según RAE: "Técnica de la indagación y del descubrimiento". "En algunas ciencias, manera de buscar la solución de un problema mediante métodos no rigurosos, como por tanteo, reglas empíricas u otras".

alta calidad utilizando técnicas y principios propios de la ingeniería. El problema ingenieril normalmente no se puede resolver con una fórmula matemática, sino que es de naturaleza iterativa. El uso de los principios y prácticas de la ingeniería permiten garantizar que los sistemas de software cumplan con las necesidades y expectativas definidas y que, además, sean cuantificables [Sawy01] [Wohl18].

3.2 Ingeniería de Software

La Ingeniería de Software (IS) [Naur69] [Boeh76] [Boeh81] [Broo95] [Shaw96] [Somm11] es una disciplina esencial en la creación de soluciones tecnológicas complejas y sostenibles. Al seguir principios y metodologías probadas, permite desarrollar software de alta calidad, confiable, eficiente, que pueda adaptarse a las necesidades cambiantes del mundo.

La Ingeniería de Software es definida de la siguiente manera:

> **Bauer** en la Conferencia de la OTAN (1969), "[La Ingeniería de Software es] *el establecimiento y uso de principios fundamentales de la ingeniería con objeto de desarrollar en forma económica software que sea confiable y que trabaje con eficiencia en máquinas reales".*
>
> **Boehm** (1976), "*La Ingeniería de Software es la aplicación práctica del conocimiento científico al diseño y construcción de programas de computadora y a la documentación asociada requerida para desarrollar, operar y mantenerlos. Se conoce también como desarrollo de software o producción de software".*
>
> **Software Engineering Institute** (1990), "*Esa forma de ingeniería que aplica los principios de la informática y las*

matemáticas para conseguir soluciones rentables a problemas software".

Pressman & Maxim (2020), "*Se trata del establecimiento de los principios y métodos de la ingeniería a fin de obtener software de modo rentable, que sea fiable y trabaje en máquinas reales".*

Algunos puntos clave que marcan la relevancia de esta disciplina en la producción de software:

- *Calidad del producto software*. Garantiza que los productos desarrollados cumplan con estándares de calidad, incluyendo aspectos como confiabilidad, mantenibilidad, escalabilidad y usabilidad.
- *Comunicación*. Proporciona enfoques estructurados para dividir proyectos grandes en módulos manejables, reduce la probabilidad de errores durante el desarrollo y facilita que los desarrolladores y los involucrados en el proyecto comprendan de la misma manera cómo será el software.
- *Menor costo de errores.* Al aplicar las buenas prácticas, se pueden reducir los costos de desarrollo mediante la prevención de errores en etapas tempranas y evitar retrasos gracias a una mejor planificación y la gestión del proyecto.
- *Mantenimiento*. Es una de las fases más largas y costosas del ciclo de vida del software. La Ingeniería de Software fomenta el diseño de sistemas que sean fáciles de actualizar para añadir nuevas funcionalidades, corregir errores y ser adaptables a cambios tecnológicos.

- *Mejora la producción de software*. Mediante el uso de herramientas, procesos y técnicas automatizadas, la Ingeniería de Software permite aumentar la velocidad del desarrollo, minimizar tareas repetitivas mediante herramientas como sistemas de control de versiones, entornos integrados de desarrollo, etc. y estandarizar procesos para que los equipos trabajen de manera coordinada.
- *Menor riesgos.* Cuenta con herramientas para identificar y mitigar riesgos técnicos, financieros y organizativos en el desarrollo de proyectos.
- *Sustentabilidad.* Promueve el uso de estándares y buenas prácticas que garantizan la sostenibilidad del software a largo plazo.
- *Flexibilidad ante el cambio*. Ayuda a que el desarrollo de software incorpore los cambios tecnológicos y metodológicos existentes sin comprometer la calidad del producto software.
- *Sustentabilidad*. Contribuye al desarrollo de sistemas que mejoren la calidad de vida de las personas, fomenten la innovación tecnológica y apoyen el crecimiento económico al permitir a las empresas ofrecer productos y servicios más eficientes.
- *Escalabilidad*. Facilita la creación de sistemas capaces de escalar y manejar mayores volúmenes de datos o usuarios.

Se puede observar que la Ingeniería de Software es un abordaje estratégico para garantizar el éxito del desarrollo de sistemas de software complejos. Proporciona un marco de trabajo eficiente, maximizando el valor del software entregado y minimizando los riesgos asociados.

Cabe destacar que no solo abarca las actividades propias de la construcción del software, entre las que se encuentra la *Ingeniería de Requisitos*, sino que incluye una variedad de actividades complementarias y transversales que son esenciales para garantizar el éxito del proyecto. Entre estas actividades se encuentra la gestión de proyectos, la gestión de la configuración, la gestión del riesgo, el aseguramiento de la calidad (SQA), el despliegue y la implementación, la capacitación y el soporte técnico, la evaluación del software, la gestión del cambio, las tareas de investigación y desarrollo (I+D), la evaluación post-implementación, entre otras.

La Ingeniería de Requisitos y la Ingeniería de Software son complementarias e interdependientes. Mientras que la Ingeniería de Requisitos define qué debe construirse y establece las bases para el desarrollo, la Ingeniería de Software proporciona los procesos, herramientas y técnicas necesarias para transformar esos requisitos en un producto funcional y de calidad. Una colaboración efectiva entre ambas es esencial para el éxito de cualquier proyecto de software.

3.3 Ingeniería de Requisitos (IR)

La Ingeniería de Requisitos [Naur69] [Davi93] [Louc95] [Wieg13] [Hull17] [Lapl17] es una disciplina que permite definir los servicios que el nuevo sistema de software proveerá y las restricciones y limitaciones que deberá contemplar. Esta fase temprana en la construcción del software se torna relevante cuando se está en presencia de un cliente que desconoce el impacto que producirá la inclusión de un sistema de software en la organización.

Aunque el problema de cómo especificar un sistema no es nuevo, la comunidad científica de computación, principalmente la de Ingeniería

de Software, debió prestar una especial atención a la tarea de definir *qué* debe hacer el software y asegurar un punto de partida confiable. Como área de estudio formal y práctica profesional se fue estableciendo gradualmente a lo largo de los años 70 y 80. El término Ingeniería de Requisitos se acuñó por primera vez en 1977 en IEEE Transactions on Software Engineering [Davi77]. En 1993, el grupo de trabajo IFIP TC-2[6] organizó conferencias y actividades relacionadas a la Ingeniería de Software, incluyendo la Ingeniería de Requisitos como una sub-disciplina clave, lo que pudo consolidar su importancia y establecerla como una de las actividades más tempranas y críticas en la construcción de software [Beat13]. En esta década ya estaba plenamente integrada al proceso de construcción del software.

La importancia de la Ingeniería de Requisitos radica en garantizar que las necesidades y expectativas de los usuarios, clientes y otras partes interesadas sean identificadas, analizadas, documentadas y gestionadas adecuadamente durante todo el ciclo de vida del proyecto. Algunas definiciones:

> **Loucopoulos** (1995), *"La Ingeniería de Requisitos puede ser definida como un proceso sistemático de desarrollo de los requisitos a través de un proceso cooperativo-interactivo de análisis del problema, documentando los resultados observados en una variedad de formatos de representación, y chequeando la comprensión obtenida. Las actividades envueltas en la Ingeniería de Requisitos ayudan a comprender las exactas necesidades de los usuarios de un sistema de software y trasladar esas*

[6] TC-2 (Technical Committee 2): es un comité técnico de IFIP (International Federation for Information Processing) que se enfoca específicamente en el campo del software. Su nombre completo, *"Software: Theory and Practice"*, aborda tanto los aspectos teóricos como prácticos relacionados con el diseño, desarrollo, implementación y mantenimiento del software.

necesidades en sentencias precisas y sin ambigüedad las que luego serán utilizadas en el desarrollo de software".

Zave (1997), "*Rama de la Ingeniería del Software que trata con el establecimiento de los objetivos, funciones y restricciones de los sistemas de software. Asimismo, se ocupa de la relación entre estos factores con el objeto de establecer especificaciones precisas".*

Sommerville (2011), *"La Ingeniería de Requisitos es un proceso que comprende todas las actividades requeridas para crear y mantener un documento de requisitos del sistema. Existen cuatro actividades genéricas de alto nivel en este proceso. Estas son el estudio de factibilidad del sistema, la obtención y análisis de los requisitos, la especificación y la documentación, y finalmente la validación".*

Laplante (2017), *"La Ingeniería de Requisitos es el proceso sistemático de obtener, modelar, analizar y validar los requisitos del software para garantizar que el sistema final cumpla con las necesidades de los usuarios y otras partes interesadas".*

Van Lamsweerde (2018), "*La Ingeniería de Requisitos abarca un conjunto de actividades organizadas para identificar y especificar lo que un sistema debe hacer, y asegurarse de que esté alineado con los objetivos del negocio y las restricciones del entorno".*

Algunos puntos que demuestran la importancia de esta fase:

- *Definición de lo que se va a construir.* La Ingeniería de Requisitos ayuda a definir y clarificar *qué debe hacer el sistema* de software antes de que comience su desarrollo. Esto incluye

identificar las necesidades del cliente, usuarios y de las partes interesadas, especificar los objetivos del sistema, funcionalidades y restricciones del software y evitar malentendidos y confusiones en etapas posteriores. Es durante la Ingeniera de Requisitos cuando se establecen las bases para todo el desarrollo del sistema de software.

- *Minimización de errores y re-trabajos futuros*. Los errores en los requisitos son una de las principales causas de fallas en los proyectos software. Al establecer requisitos claros y bien documentados se reducen los cambios inesperados durante el desarrollo y se evita el re-trabajo costoso en fases posteriores.
- *Aseguramiento de la calidad del producto*. La Ingeniería de Requisitos garantiza que el producto final cumpla con los estándares de calidad, que las funcionalidades entregadas sean útiles y relevantes para los usuarios y que el software esté alineado a los objetivos organizacionales.
- *Mejoramiento de la comunicación y la colaboración*. Garantiza que los todos los que participan en la construcción del software (clientes, usuarios, interesados, desarrolladores) cuenten con una comunicación eficaz y asegurar que todos comprenden de igual manera los requisitos del software.
- *Determinación del alcance del proyecto.* La Ingeniería de Requisitos define el alcance del proyecto, lo que ayuda a evitar la inclusión de características innecesarias (scope creep), gestionar las expectativas de las partes interesadas y establecer límites claros para el trabajo de todo el equipo de desarrollo.
- *Optimización de tiempos y costos del software*. Un enfoque riguroso en la Ingeniería de Requisitos evita desperdiciar

tiempo y recursos en desarrollar características que no son necesarias o que no cumplen con las necesidades del cliente. Esto resulta en un mejor control sobre los costos del proyecto y un uso más eficiente del tiempo del equipo.

- *Soporte para pruebas y validación.* Los requisitos sirven como base para las pruebas y la validación del sistema. Permiten verificar en cualquier fase del desarrollo si se cumple con los requisitos establecidos y validar si el sistema satisface las necesidades del usuario final.
- *Adaptación a los cambios en los requisitos.* Dado que los requisitos pueden evolucionar durante el ciclo de vida del proyecto, la Ingeniería de Requisitos incluye técnicas para gestionar cambios de manera eficiente. Esto asegura que los cambios se evalúen adecuadamente en términos de impacto y que el equipo pueda responder a nuevas necesidades sin comprometer el proyecto.

La Ingeniería de Requisitos es fundamental para garantizar que el software desarrollado sea útil, viable, y cumpla con los objetivos del cliente y esté alineado a las necesidades del negocio. Cuando estos objetivos no se pueden alcanzar, las consecuencias pueden ser muy graves, como un producto que no es útil para el cliente, incremento en los costos, retrasos, proyectos fallidos o sistemas inutilizables. Algunos de estos problemas se relacionan con la gran diversidad de enfoques existentes para construir sistemas de software, ya que introduce variabilidad en cómo se obtiene la información del contexto y se gestionan y validan los requisitos. Los ingenieros/as de requisitos deben ser flexibles y adaptarse a las metodologías y perspectivas que están presentes en cada proyecto, equilibrando la necesidad de

estabilidad con la capacidad de adaptación al cambio. Para ejemplificar este pensamiento, a continuación, se describen brevemente algunos enfoques.

En 1977, Chuck Morris y Tony Crawford de IBM desarrollaron un método para determinar los requisitos de un sistema de información basado en técnicas de talleres conocido como ***Diseño de Aplicación Conjunta*** (JAD). En estos talleres o reuniones el interés estaba en las personas, enfatizando el desarrollo participativo entre el cliente, los usuarios y los desarrolladores. Para Wood et al. [Wood95] JAD permite diseñar sistemas de calidad en un 40% menos de tiempo.

Como ya se mencionó, en 1987 Mills et al. publicó el método de ***Cleanroom*** [Mill87] con el propósito de reducir a cero la ambigüedad y maximizar la precisión del documento de especificación. Este enfoque puede asegurar que el sistema de software satisface los requisitos, pero no aseguran que satisfacen las necesidades del cliente.

Wirfs-Brock et al. propusieron el ***Método Dirigido por Responsabilidades*** [Wirf95], que es una metodología de análisis y diseño orientada a objetos (ADOO). Esta se diferencia de las metodologías tradicionales orientadas a objetos en que analiza el contexto donde viven los objetos, mientras que las demás analizan directamente los objetos.

Se puede observar que en los métodos descriptos existe una clara tendencia a asegurar los servicios del software más que concentrarse en satisfacer las necesidades reales del cliente.

Yu et al. propusieron el ***enfoque i**** [Yu93] [Yu95] el cual realiza diferentes modelos orientados a comprender la organización y los objetivos del negocio para definir requisitos que soporten esos

objetivos, para asignar responsabilidades en el sistema, para resolver conflictos de puntos de vista, etc.

Diferentes autores [Carr95] [Zorm95] [Roll98] [Weid98] [Leit00] proponen el uso de ***escenarios***, los cuales describen situaciones del contexto. En este modelo, los requisitos del software están empotrados, por lo tanto, pueden ser incluidos directamente en un documento de requisitos o generar un documento diferente. Una ventaja de este modelo es que utiliza el lenguaje natural para su representación, lo que permite la colaboración y participación de los involucrados/as durante toda la Ingeniería de Requisitos.

En los métodos recién descriptos la tendencia se modifica y se concentran más en satisfacer las necesidades reales del cliente, objetivo fundamental de la Ingeniería de Requisitos.

Referencias

[Baue69] Bauer F. L., Bolliet L., Helms H. J. (1969). "Software Engineering". Report on a conference sponsored by the NATO SCIENCE COMMITTEE Garmisch, Germany, 7th to 11th October 1968. Editors: Peter Naur and Brian Randell

[Beat13] Beatty, J. (2013). "Software requirements" (3rd ed.). Microsoft Press.

[Boeh76] Boehm, B.W. (1976). "Software Engineering", IEEE Transaction computers, Vol.25, Nº12, pp.1226-1241

[Boeh81] Boehm, B. W. (1981). "Software Engineering Economics". Prentice Hall.

[Broo95] Brooks, F. P. (1995). The Mythical Man-Month: Essays on Software Engineering (20th Anniversary Edition). Addison-Wesley.

[Carr95] Carroll, J., "Introduction: The Scenario Perspective on System Development" (1995). En el libro Scenario-Based Design: Envisioning Work and Technology in System Development, editor J. Carroll, John Wiley & Sons, Nueva York, pp.1-18.

[Davi77] Davis, A. M. (1977). "Requirements engineering: A roadmap". IEEE Transactions on Software Engineering, SE-3(1), 1-10. https://doi.org/10.1109/TSE.1977.234101

[Davi93] Davis, A. M. (1993). "Software requirements: Objects, functions, and states". Prentice Hall.

[Hull17] Hull, E., Jackson, K., & Dick, J. (2017). "Requirements engineering" (4th ed.). Springer.

[Lapl17] Laplante, P. A. (2017). "Requirements Engineering for Software and Systems" (3rd ed.). CRC Press.

[Leit00] Leite, J.C.S.P., Hadad, G.D.S., Doorn, J.H., Kaplan, G.N. (2000). "Scenario Construction Process", Requirements Engineering Journal, Springer-Verlag London Ltd., Vol.5, Nº1, pp. 38-61.

[Louc95] Loucopoulos, P., Karakostas, V. (1995). "System Requirements Engineering", McGraw-Hill, Londres.

[Mill87] Mills, Harlan D.; Dyer, M.; and Linger, R. C. (1987). "Cleanroom Software Engineering". The Harlan D. Mills Collection.

[Naur69] Naur, P., & Randell, B. (Eds.). (1969). "Software Engineering: Report on a Conference Sponsored by the NATO Science Committee". Garmisch, Germany.

[Pres20] Pressman, R. S., & Maxim, B. R. (2020). "Software engineering: A practitioner's approach" (9th ed.). McGraw-Hill Education.

[Somm11] Sommerville, I. (2011). "Ingeniería de Software", Pearson, Edition 9.

[Roll98] Rolland, C., Souveyet, C, Ben Achour, C. (1998). "Guiding Goal Modeling Using Scenarios", IEEE TSE, Vol.24, N°12, pp.1055–1071.

[Sawy01] Sawyer, P., Kotonya, G. (2001). "Software Requirements", SWEBOK, Guide to the Software Engineering Body of Knowledge, editores P. Bourque y R. Dupuis, IEEE Computer Society, Los Alamitos, CA, Capítulo 2, pp.31-56, http://www.swebok.org/stoneman/ trial_1_00.html.

[VanL18] Van Lamsweerde, A. (2018). "Requirements Engineering: From System Goals to UML Models to Software Specifications". (2nd ed.). Wiley.

[Weid98] Weidenhaupt, K., Pohl, K., Jarke, M., Haumer, P. (1998). "Scenarios in System Development: Current Practice", IEEE Software, pp.34- 45.

[Wieg13] Wiegers, K. E., & Beatty, J. (2013). "Software requirements". (3rd ed.). Microsoft Press.

[Wirf95] Wirfs-Brock, R. (1995). "Designing Objects and Their Interactions: A Brief Look at Responsibility-Driven Design", Scenario-Based Design: Envisioning Work and Technology in System Development, editor J. Carroll, John Wiley, Nueva York, pp. 337-359.

[Wohl18] Wohlrab R., Pelliccione P., Knauss E., Gregory S.C. (2018). "The Problem of Consolidating Requirements Engineering Practices at Scale: An Ethnographic Study". In: Kamsties E., Horkoff J., Dalpiaz F. (eds) Requirements Engineering: Foundation for Software.

[Wood95] Wood, J. and D. Silver (1995). "Joint Application Development", Wiley and Sons, Inc. New York.

[Yu93] Yu, E. (1993). "Modeling organizations for information systems requirements engineering", IEEE First International Symposium on Requirements Engineering, IEEE Computer Society Press, San Diego, pp.34-41.

[Yu95] Yu, E. (1995). "Modeling Strategic Relationships for Process Reengineering", Ph.D. thesis, Department of Computer Science, University of Toronto, Canada.

[Zorm95] Zorman, L. (1995). "Requirements Envisaging by Utilizing Scenarios (Rebus)", Ph.D. Dissertation, University of Southern California.

Capítulo

4 Análisis de Sistemas e Ingeniería de Requisitos

A pesar del tiempo transcurrido y de toda la actividad académica y profesional referida al desarrollo de sistemas de software y particularmente a la fase de construcción de los requisitos del software, todavía es difícil encontrar una definición clara y precisa que establezca los límites y las diferencias entre el Análisis de Sistemas (AS) [DeMo78] [Gane79] [Your88] [Seen91] [Whit04] [Hoff14] [Kend19] y la Ingeniería de Requisitos (IR). Si bien ambas tienen el mismo objeto de estudio y el mismo objetivo, utilizan diferentes puntos de vista y diferentes formas de representación. Diferentes puntos de vista porque en el Análisis de Sistemas se atienden, por ejemplo, los datos o las entidades participantes, mientras que la Ingeniería de Requisitos presta atención a objetivos, conductas y necesidades. Diferentes formas de representación, porque el Análisis de Sistemas utiliza modelos cercanos al diseño y la Ingeniería de Requisitos utiliza modelos cercanos al cliente. De esta manera el Análisis de Sistemas privilegia la especificación de los servicios del sistema a ser desarrollado utilizando modelos o construcciones que faciliten la comprensión por parte de los desarrolladores. Esto simplifica enormemente la actividad del diseño, pero mutila la comprensión del cliente inhibiendo la percepción de

todas las consecuencias que ocurrirán cuando el sistema de software se ponga en producción. Por otro lado, la Ingeniería de Requisitos privilegia la especificación de los servicios del sistema a ser desarrollado utilizando modelos o construcciones que faciliten la comprensión por parte del cliente. Esto dificulta en forma sensible la actividad de diseño. Naturalmente que son posibles infinidad de situaciones intermedias, pero siempre existirá la inexorable tensión entre facilitar un punto de vista o el otro.

4.1 Perspectiva del Análisis de Sistemas

Desde una mirada más metodológica que estratégica, se puede observar que el Análisis de Sistemas ha tomado poca distancia del diseño y del código, provocando una suerte de simbiosis metodológica en pos de una mayor calidad en el producto software a construir, calidad entendida en el contexto del sistema en sí propio, es decir procurando obtener un producto robusto y fácil de mantener. En las siguientes definiciones se puede ver reflejado lo antes dicho:

> **Senn** (1991), *"El Análisis de Sistemas es el proceso de clasificación e interpretación de hechos, diagnóstico de problemas y empleo de la información para recomendar mejoras al sistema. Este es el trabajo del analista de sistemas".*
>
> **Kendal & Kendal** (1997), "*El análisis y diseño de sistemas que los analistas de sistemas llevan a cabo busca comprender qué necesitan los humanos para analizar la entrada o el flujo de datos de manera sistemática, procesar o transformar los datos, almacenarlos y producir información en el contexto de una organización específica. Mediante un análisis detallado, los analistas buscan identificar y resolver los problemas correctos.*

> *Además, el análisis y diseño de sistemas se utiliza para analizar, diseñar e implementar las mejoras en el apoyo para los usuarios y las funciones de negocios que se puedan llevar a cabo mediante el uso de sistemas de información computarizados".*

El Análisis de Sistemas expresa los requisitos en términos del producto "el sistema hará un reporte mensual con las ventas que superen las 1000 unidades". En estos términos la construcción de los sistemas de software se direcciona más al producto que a las necesidades del cliente, o dicho de otra manera es probable que algunas decisiones de diseño no estén totalmente alineadas a las necesidades del cliente y sea este último quien deba adaptarse a alguna funcionalidad del software. Los requisitos del software en el Análisis de Sistemas describen el producto software en función a los datos, las funciones, los procesos del software, etc. A diferencia del AS, la Ingeniería de Requisitos describe los servicios que tendrá el nuevo sistema de software desde el punto de vista del cliente.

4.2 Perspectiva de la Ingeniería de Requisitos

Como se mencionó en el capítulo anterior, la Ingeniería de Requisitos comienza cuando el cliente detecta una necesidad que potencialmente se puede satisfacer con un sistema informático. Desde ese momento se empiezan a gestar los requerimientos que a través de un proceso de refinamiento permiten dilucidar la mejor propuesta para el negocio, o sea los requisitos del software. Cabe destacar que, a diferencia del Análisis de Sistemas, estos requisitos no incluyen aspectos de diseño. Esto le permite a la Ingeniería de Requisitos identificar los servicios que responden a las necesidades reales del cliente desde la mirada del negocio. Por lo tanto, la Ingeniería de Requisitos es responsable de

generar una especificación de requisitos junto al cliente con el objetivo de construir la solución óptima para la organización que solicitó el sistema de software.

4.3 Diferencias entre el Análisis de Sistemas y la Ingeniería de Requisitos

Puede observarse que las definiciones de Análisis de Sistemas y de Ingeniería de Requisitos prácticamente coinciden en sus objetivos, sin embargo, como ya se describió, es en la forma en que se concretan las mismas donde reside la gran diferencia. Efectivamente, establecer con mayor precisión esta diferencia requiere abandonar el nivel de objetivos para avanzar sobre las materializaciones que se hicieron de los mismos.
El Análisis de Sistemas produce una descripción del sistema de software en términos técnicos esencialmente computacionales, mientras que la Ingeniería de Requisitos lo hace en términos de los servicios perceptibles por el cliente.

	Calidad de la definición de los servicios del software	**Facilidad para el diseño**
AS	Pobre	Buena
IR	Buena	Pobre

Tabla 3 - Ponderación de la distancia desde la IR y el AS al diseño

En la Tabla 3 se describen estas características que marcan y definen el comportamiento de cada uno, ya que para la discusión acerca de las ventajas relativas del Análisis de Sistemas sobre la Ingeniería de Requisitos o viceversa, se debe ponderar la importancia relativa de la

calidad de la definición de los servicios y la facilidad para el diseño del sistema.

Como se mencionó, el resultado producido por el Análisis de Sistemas dificulta la percepción del cliente del grado de satisfacción de sus expectativas, pero facilita la tarea de diseñar el sistema. Por su parte el resultado producido por la Ingeniería de Requisitos facilita la precepción del cliente, por lo que contribuye a una mejor definición de los servicios que tendrá el software, pero paga el precio de dificultar el diseño posterior del sistema.

Puede observarse en la Figura 8 que la salida esperada del Análisis de Sistemas se constituye en un conjunto de esquemas, gráficos, diagramas que abstraen el proceso del negocio en conceptos computacionales muy distantes al cliente, lo que dificulta su comprensión. Mientras que la Ingeniería de Requisitos propone modelos lo más cercano al contexto en estudio, donde el aprendizaje para comprender la propuesta se reduzca a la mínima expresión. Este es el motivo central por el cual, aproximadamente el 70% de los modelos utilizados, representan toda la información en lenguaje natural.

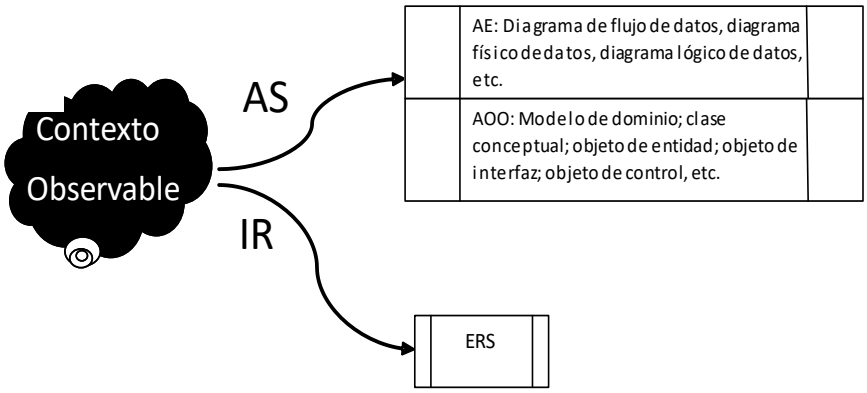


Figura 8 - Artefactos generados por el AS y la IR

Esta cercanía al punto de vista del software le permite al Análisis de Sistemas detectar rápidamente la viabilidad de los pedidos del cliente. Algo que a la Ingeniería de Requisitos le resulta más difícil de percibir. Por lo tanto, cuando un cliente solicita un servicio o una restricción, el equipo de requisitos debe estar atento a que ese pedido pueda ser incluido, ya que de no ser posible o genere costos excesivos, se convertirá en un problema al avanzar con la construcción del software. Esto lo obliga a estar actualizado para reconocer la viabilidad de los servicios posibles al momento de estudiar cada problema en particular.

El Análisis de Sistemas procura describir la realidad observable utilizando lo más temprano posible técnicas y modelo propios del procesamiento de datos, procurando planificar el proceso del negocio futuro en esos términos, mientras que la Ingeniería de Requisitos procura describir la realidad observable utilizando el lenguaje y los puntos de vista del cliente; planificando el proceso del negocio futuro también en términos del punto de vista de los involucrados/as. En la Figura 9 se puede observar que el Análisis de Sistemas genera diagramas que rápidamente se trasladan al diseño, mientras que la Ingeniería de Requisitos está más lejos y necesita un esfuerzo adicional para lograrlo.

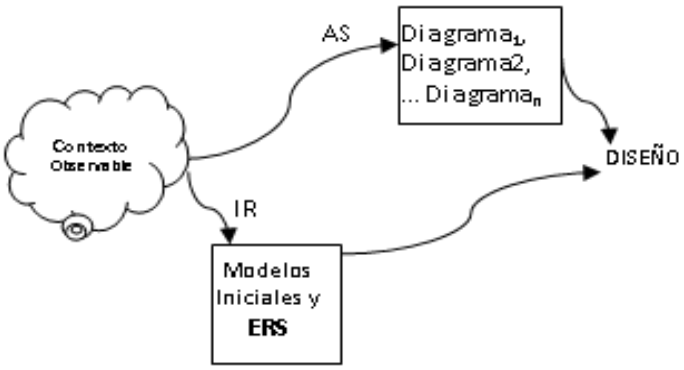


Figura 9 - Distancia al diseño desde el AS y la IR

En la Figura 9 se identifican dos distancias cognitivas. Cabe aclarar que esta distancia incluye solo la comprensión y el esfuerzo tanto del cliente como del desarrollador/a. La primera distancia corresponde al camino entre el mundo observable y la obtención de los requisitos del software. La segunda, es el salto hacia el diseño.

Cabe destacar que tanto la Figura 8 como la Figura 9 se han pensado en un entorno de la aplicación de un Modelo de Proceso en Cascada y que deben ser reinterpretadas dependiendo del modelo de proceso que se trate.

En resumen, el precio que se paga por ir por el camino del Análisis de Sistemas es más caro porque no asegura los resultados y eso repercute en la calidad y viabilidad del producto. Es por eso que la comunidad de informática ha evolucionado del Análisis de Sistemas a la Ingeniería de Requisitos procurando un acercamiento al contexto y a las necesidades reales del cliente.

Referencias

[DeMa78] DeMarco, T. (1978). "Structured Analysis and System Specification" Yourdon. ISBN 978-0-917072-07-9.

[Gane79] Gane, C., & Sarson, T. (1979). "Structured Systems Analysis: Tools and Techniques". Prentice Hall.

[Hoff14] Hoffer, J. A., George, J. F., & Valacich, J. S. (2014). "Modern systems analysis and design" (7th ed.). Pearson.

[Kend19] Kendall, K. E., & Kendall, J. E. (2019). "Systems analysis and design" (10th ed.). Pearson.

[Senn91] Senn J. (1991). "Análisis y Diseño de Sistemas de Información", Segunda Edicion, McGraw Hill.

[Whit04] Whitten, J. L., Bentley, L., & Dittman, K. C. (2004). "Systems analysis and design methods" (7th ed.). McGraw-Hill Education.

[Your88] Yourdon, E. (1988). “Modern Structured Analysis”, Prentice Hall; Edición 1, ISBN-10: 0135986249, ISBN-13: 978-0135986240.

Capítulo 5

Roles de la Ingeniería de Requisitos

La Ingeniería de Requisitos requiere de un trabajo colaborativo donde participan básicamente dos grupos bien definidos: involucrados/as y desarrolladores/as. El primero representa a la organización, mientras que el segundo a quienes van a construir el software, en este caso particular a los que tienen relación con la Ingeniería de Requisitos. Dentro de cada grupo se dan diferentes roles, siendo los más representativos del grupo involucrados/as el rol *cliente* y *usuario* y dentro de los desarrolladores/as el rol *ingeniero de requisitos*, *gestor del proyecto* y *equipo de calidad* o *QA*. Cabe destacar que pueden existir otros roles como el equipo de seguridad, consultores especializados, etc. dependiendo de las necesidades de cada proyecto.

Es importante aclarar que desde este momento en adelante cuando se mencione al ingeniero de requisitos se está haciendo referencia al rol y no a la persona. Lo mismo ocurre con cliente-usuario, gestor de proyecto y QA.

En la Figura 10 se puede observar que el ingeniero de requisitos interactúa con el cliente-usuario para asegurar una buena colaboración y participación. También interactúa con el gestor del proyecto para tomar decisiones en cuanto al avance del proyecto y con QA para realizar las verificaciones y asegurar que todo lo construido cumple con

lo esperado por el cliente. El equipo QA también se asegura que se cumplan los aspectos de calidad establecidos (documentación, normas, etc.).

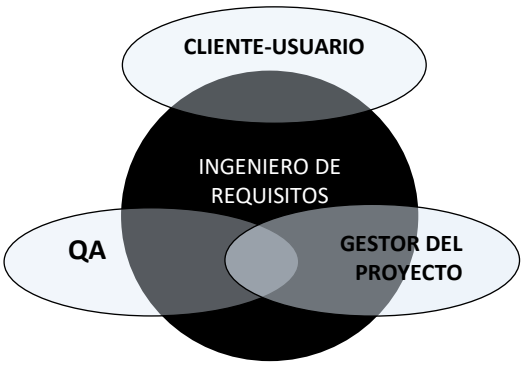


Figura 10 – Roles de la Ingeniería de Requisitos

5.1 Grupo Involucrados/as

Involucrados (stakeholders) es un término muy utilizado en la Ingeniería de Software (IS) [Shar99] [Gonz10], pero su origen es del área de la Administración. Por su uso tan diverso, existen muchos homónimos en diferentes contextos y en cada caso se incluyen grupos particulares de personas, siendo en algunos casos muy sutil la diferencia. Por tal motivo, se presenta una breve historia del uso del término. El Instituto de Investigación de Stanford introdujo el término stakeholder por primera vez en la literatura en 1963 [Free01] para referirse a un reducido grupo de personas estrechamente comprometidas con el funcionamiento de la empresa, básicamente: accionistas, empleados, clientes y aquellos con capacidades técnicas esenciales para las operaciones de la compañía. Fue así como, en un primer momento, los stakeholders se identificaron con "aquellos grupos sin cuya contribución la corporación dejaría de existir". Estos stakeholders podían no participar en la propiedad de la empresa, pero eran fundamentales para la planificación empresarial y el éxito de la

firma. Siguiendo esta línea, Freeman define stakeholder como "todo grupo o individuo que puede afectar o verse afectado por la consecución de los objetivos de la organización".

En el contexto de las actividades apoyadas por el Banco Mundial, los stakeholders son "los afectados por el resultado, negativo o positivamente o que puedan afectar el resultado de una intervención propuesta" "Los stakeholders pueden incluir: Prestatarios (funcionarios electos, personal de línea de agencia, los funcionarios del gobierno local, y así sucesivamente), los grupos directamente afectados (incluyendo pobres y desfavorecidos), los grupos afectados indirectamente (por ejemplo, las ONG y organizaciones del sector privado), y la administración del Banco Mundial, el personal y los accionistas" [ESD98].

El Business Dictionary define stakeholders como "una persona, grupo u organización que tenga interés directo o indirecto en una organización, ya que puede afectar o verse afectados por las acciones de la organización, los objetivos y las políticas. Las partes interesadas en una organización empresarial incluyen los acreedores, clientes, directores, empleados, gobierno (y sus agencias), los propietarios (accionistas), proveedores, sindicatos y la comunidad de la cual la empresa obtiene sus recursos".[7]

Sternberg en [Ster99] describe la responsabilidad de los interesados/as como una participación consiente y define a los involucrados/as como "aquellos en quienes la organización tiene un interés". Por otro lado, Alonso et al. afirman que es los involucrados/as son "todos/as aquellos/as que tienen un interés en la organización" [Alon08].

[7] http://www.businessdictionary.com/definition/stakeholder.html

En el campo de la Ingeniería de Software existe consenso acerca de que los involucrados/as son aquellas personas que tienen un interés directo o indirecto con el nuevo sistema de software y que proporcionan la información necesaria sobre las expectativas, necesidades, limitaciones y objetivos del sistema. Pero también, existen diferentes puntos de vista en cuanto a si este grupo incluye o no al grupo desarrolladores. A continuación, se presentan algunas definiciones:

> **Leite** (1997), *"las personas involucradas en el proceso de desarrollo son principalmente: usuarios, clientes, ingenieros de software, expertos del dominio, las cuales son denominadas en la literatura con la palabra 'stakeholders'"*
>
> **Pouloudi** (1999), "*son los participantes en el desarrollo de procesos junto con cualquier otro individuo, grupos u organizaciones cuyas acciones pueden influir o ser influido por el desarrollo y uso del sistema ya sea directa o indirectamente*".
>
> **Sommerville** (2011), *"todos aquellos que se benefician en una forma directa o indirecta del sistema que está en desarrollo"*. Entre los involucrados/as se encuentran las/los usuarios finales que interactúan con el sistema y todas las personas en la organización que se pueden ver afectadas por su instalación. Otros involucrados/as del sistema pueden ser los ingenieros/as que desarrollan o dan mantenimiento a otros sistemas relacionados, los/las gerentes del negocio, los expertos/as en el dominio del sistema y los representantes de los trabajadores/as.

La primera definición incluye al grupo desarrolladores/as mientras que las otras dos definiciones, no lo hacen. En el presente libro cuando se menciona a los involucrados/as se lo utiliza en el marco de la definición

de Sommerville. Como roles activos dentro del grupo involucrados/as solo se diferenciará el rol cliente, usuario y cliente-usuario.

5.1.1 Rol cliente

El rol *cliente* es quien demanda el nuevo sistema de software y se ocupa de definir el *objetivo del sistema*. También participa en la validación, aceptación y en la priorización de los requisitos. En el caso de la priorización, su participación es para aquellos requisitos que no se pueden cumplir y, por lo tanto, ayuda a priorizar qué características son más importantes para el negocio o el usuario.

Se debe considerar que el cliente suele expresarse en un nivel de abstracción alto. Esto se debe a que, por lo general, el rol cliente lo realiza una persona con un cargo jerárquico dentro de la organización (director, gerente o jefe).

5.1.2 Rol usuario

El rol *usuario* es quien se ocupa de los detalles operativos y se expresa en términos más concretos. Dentro del rol usuario es necesario identificar quienes son las personas que pueden aportar conocimiento para la Ingeniería de Requisitos. En contextos donde no existe una forma obvia de determinar a quién se debe entrevistar o existe un riesgo de dejar afuera personas con información relevante, se sugiere completar un mapa de usuarios (ver Figura 11) que puede tener su origen en un organigrama, e incorporar información acerca del conocimiento o experiencia de las personas en relación a las tareas que realiza o al proceso de negocio en estudio. Se debe considerar que seleccionar las personas que realmente aportan información relevante

para la Ingeniería de Requisitos reduce significativamente el tiempo y el esfuerzo.

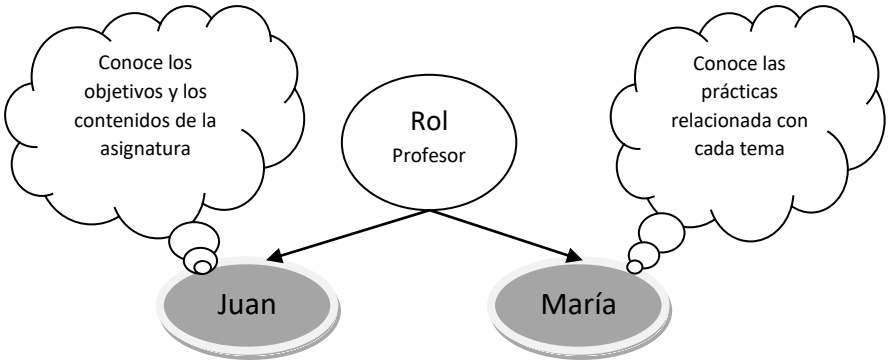


Figura 11 - Ejemplo de mapa de relación entre los roles

Es necesario mencionar que en muchas ocasiones se utiliza el rol usuario para dos funciones diferentes. Por un lado, como ya se mencionó, para identificar las personas que aportan información a la Ingeniería de Requisitos y, por otro lado, se refiere a las personas o grupo que operarán o interactuarán directamente con el producto software, o sea, el usuario final. Para evitar malos entendidos, es preferible diferenciar a qué tipo de usuario se está haciendo referencia en cada caso.

5.1.3 Rol cliente-usuario

En ocasiones el cliente y el usuario son la misma persona. Otras veces, son personas diferentes. En ambos casos estos roles pueden ser intercambiables [IEEE830]. Esto sucede cuando el cliente menciona detalles del proceso del negocio o cuando el usuario justifica algún aspecto referido con el nuevo sistema de software. Con el objetivo de evidenciar este intercambio de roles, que suele ser transparente para el ingeniero de requisitos, se utiliza el rol *cliente-usuario*.

5.2 Grupo Desarrolladores/as

En este grupo están los/las responsables de definir los requisitos del software y de gestionarlos adecuadamente. Se encuentra los siguientes roles: ingeniero de requisitos, gestor de proyectos y QA. Este grupo se conforma en relación a la necesidad de cada proyecto. Incluir diferentes perfiles asegura una mejor cobertura de todas las necesidades que puedan surgir. Esta división de tareas es importante porque mejora la especialización y el enfoque, ya que cada uno se concentra en un objetivo particular.

5.2.1 Rol ingeniero de requisitos

Este rol puede ser realizado por una persona o varias, dependiendo del proyecto. Tiene la responsabilidad del resultado de la Ingeniería de Requisitos, facilita la comunicación durante todo el proceso, asegura que los requisitos reflejan correctamente las necesidades del cliente y es responsable de garantizar los cambios en los requisitos. Se encarga de la gestión continua de los requisitos, de la priorización, la trazabilidad de los requisitos, el control de versiones, la mitigación de riesgos y de facilitar la comunicación entre todas las partes. Su objetivo principal es minimizar el impacto de los cambios en los requisitos durante todo el desarrollo del software (*gestión del cambio*) y asegurar que estos sean factibles dentro del alcance del proyecto.

5.2.2 Rol gestor del proyecto

El rol gestor de proyecto supervisa la planificación, ejecución y seguimiento del proyecto. Asegura que se cumplan los plazos y los presupuestos establecidos. Colabora con el ingeniero de requisitos para alinear los requisitos con los objetivos del proyecto, asegurándose de

que estos se prioricen de acuerdo a las restricciones de tiempo y recursos. También ayuda en la toma de decisiones si hay conflictos entre las necesidades del cliente y el equipo de desarrollo.

5.2.3 Rol QA

Lo más habitual es contar con un equipo *QA proveedor* (aseguramiento de la calidad del grupo desarrolladores/as). Su responsabilidad es garantizar la calidad de los requisitos, se encarga de revisar y auditar las actividades, el producto software y asegurar que se cumplen todas las normativas y estándares de calidad. Entre las actividades en las que suele participar se encuentran: verificar los requisitos, definir criterios de aceptación, construcción del modelo de trazabilidad, identificación de riesgos y planificación de casos de pruebas. Cuando existe un equipo QA cliente (aseguramiento de la calidad del grupo involucrados/as) se involucra en las etapas iniciales de la construcción del software para garantizar que los requisitos satisfagan las necesidades del negocio y que se puedan probar de manera efectiva al finalizar el proyecto. Finalmente, cuando existe un equipo QA en ambas partes, *QA proveedor-cliente*, colaboran estrechamente, mientras que QA proveedor se centra en las pruebas técnicas y funcionales, QA cliente puede centrarse en pruebas de aceptación y validación desde la perspectiva del negocio.

5.3 Relaciones de poder cliente-proveedor

La relación de poder dentro de la Ingeniería de Requisitos se da entre el *cliente* del grupo involucrados/as y el *proveedor* del software. El cliente y el proveedor interactúan fuertemente durante todo el proceso y es notable la forma en que se ve afectado el mismo por la relación de

poder que pueden ejercer. En otras palabras, como se puede observar en la Figura 12, se producen resultados muy diferentes si se está en presencia de un grupo cliente dominante, un proveedor dominante o una relación de poder equilibrada.

Figura 12 - Tipos de relación cliente - proveedor

La situación de *Cliente dominante* ocurre cuando en la organización existe un área de desarrollo de software que depende jerárquicamente del área adonde pertenece el cliente. Por ejemplo, un departamento de desarrollo que es parte de una gerencia de comercialización o de una gerencia contable o administrativa. También ocurren situaciones del *grupo cliente dominante* cuando existen numerosos proveedores potenciales y la organización cliente tiene muchas opciones para seleccionar uno de ellos (situación de sobreoferta). Cuando ocurre alguna de estas situaciones resulta difícil o prácticamente inviable realizar una Ingeniería de Requisitos, por lo tanto, es mejor utilizar modelos de procesos aptos para desenvolverse en contextos de alta volatilidad de los requisitos, como ser procesos de prototipado o metodologías ágiles.

Situaciones de *Proveedor dominante* se producen cuando el nivel jerárquico del proveedor es más alto que el del cliente. Con proveedores externos, el caso de *proveedor dominante* se produce cuando existe sobredemanda de desarrollos. Cuando se está en

presencia de alguna de estas situaciones se debe poner especial énfasis en atender apropiadamente las necesidades del cliente-usuario de manera de evitar que la mayoría de las decisiones sean tomadas unilateralmente por el ingeniero/a de requisitos.

Una situación *equilibrada* tiene lugar cuando el cliente y el proveedor tienen una ubicación jerárquica similar en el organigrama de la organización. En el caso de proveedores externos, esta situación ocurre cuando la oferta y la demanda están equilibradas. Esta es la situación óptima para la Ingeniería de Requisitos.

Referencias

[Alon08] Pablo de Andrés Alonso, Valentín Azofra Palenzuela. (2008). "El enfoque multi stakeholder de la responsabilidad social corporativa: De la ambigüedad conceptual a la coacción y al intervencionismo". Revue Sciences de Gestion - Management Science - Ciencias de Gestión, ISSN:1634-7056), nº 66, pp. 69-90.

[ESD98] "Participation and Social Assessment: Tools and Techniques". Manufactured in the United States of America, ISBN 0-8213-4186-3, pp 4-5, 1998.

[Free01] Freeman, R. & Mcvea, John. (2001). "A Stakeholder Approach to Strategic Management", SSRN Electronic Journal. 10.2139/ssrn.263511.

[Gonz10] Carlos Hernán González Campo. (2010). "Stakeholders: una aplicación de la teoría de los stakeholdes a los negocios electrónicos", Estudios Gerenciales Print version ISSN 0123-5923 estud.gerenc. vol.26 no.114.

[IEEE830] IEEE Std 830. (1998). "IEEE Recommended Practice for Software Requirements Specifications (ANSI)", IEEE. https://ieeexplore.ieee.org/document/720574

[Leit97] Leite, J.C.S.d.P., Rossi, G., Balaguer, F. et al. Enhancing a requirements baseline with scenarios. Requirements Eng 2, 184–198 (1997). https://doi.org/10.1007/BF02745371

[Poul99] Pouloudi, A. (1999). "Aspects of the stakeholder concept and their implications for information systems development", Proceedings 32nd Annual Hawaii International Conference on System Sciences (HICSS'99).

[Shar99] Sharp, H., Finkelstein, A. & Galal, G. (1999). "Stakeholder Identification in the Requirements Engineering Process", pp 1, http://eprints.ucl.ac.uk/744/1/1.7_stake.pdf

[Somm11] Sommerville, I. (2011). "Ingeniería de Software", Pearson, Edition 9.

[Ster99] Sternberg, E. (1999). "The Stakeholder Concept: A Mistaken Doctrine. Foundation for Business Responsibilities", Issue Paper No. 4, Available at SSRN: https://ssrn.com/abstract=263144 or http://dx.doi.org/10.2139/ssrn.263144

Capítulo

6 Contexto de los requisitos

El contexto de la Ingeniería de Requisitos influye sobre la forma de definir los requisitos del software. Ya sea que se obtenga muy rápidamente un conjunto de requisitos relativamente incompleto, posiblemente dudosos y con eventuales contradicciones o que se obtengan requisitos de muy alta calidad, en un plazo apropiado. Por lo tanto, se puede definir al contexto como *todos los factores que afectan la forma de trabajar del ingeniero/a de requisitos*. Se puede observar que es un concepto ambiguo ya que no se sabe si se refiere a todo aquello que rodea a las actividades del proceso de requisitos o si se refiere a todo aquello que rodea al sistema de software a desarrollar. Por otro lado, la mayoría de los enfoques de la Ingeniería de Requisitos se desarrollan en un contexto implícito, donde no se toman en cuenta aspectos que condicionan la obtención de los requisitos del software, como ser:

- Costo de oportunidad del sistema de software.
- Poco interés de los usuarios en el nuevo sistema de software.
- Disponibilidad limitada del cliente-usuario.
- Imposibilidad para obtener información del contexto.
- Falta de políticas organizacionales claras.
- Acciones erráticas de los usuarios (por ejemplo, resolución de

problemas que entran en conflicto con reglas del negocio o normativas).

- Ausencia de procedimientos bien definidos.
- Presiones de la competencia.

Se debe tener en cuenta que la Ingeniería de Requisitos cambia cuando algunos de estos aspectos se hacen presentes, ya que el contexto determina qué proceso de requisitos se debe utilizar y define qué actividades de ese proceso se deben realizar y cuáles no.

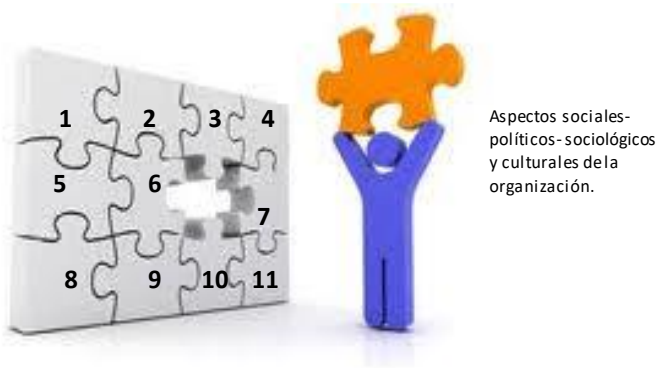
6.1 Universo de discurso

El contexto específico para el desarrollo del nuevo sistema de software se denomina Universo de Discurso (UdeD). Es importante comprender "*donde*" las actividades de la Ingeniería de Requisitos tendrán lugar y de qué manera se deben situar, atendiendo, no solo al proceso de construcción del software sino también al ambiente en donde el software se ejecutará. Algunos autores afirman que el UdeD está compuesto por el grupo de personas interesadas en comunicarse y en los artefactos de requisitos y diseño. Determinan un único UdeD para todo el proceso de desarrollo [Bjør06]. Mientras que otros autores definen UdeD como el espacio organizacional donde reside toda la información necesaria para construir el nuevo sistema de software.

> **Jackson** (1995), *"Es todo el contexto en el cual el software se desarrolla e incluye todas las fuentes de información. Es la realidad acotada por el conjunto de objetivos establecidos por quienes demandan una solución de software*"

En el UdeD conviven una gran cantidad de variables (ver Figura 13). Parte de la complejidad de la Ingeniería de Requisitos consiste en esta

convivencia que se debe contemplar a la hora de especificar los requisitos del software.



1- proceso del negocio actual; 2 - políticas organizacionales; 3 - restricciones a los procesos; 4 - información del dominio; 5 - proceso de construcción del software; 6 - necesidades del cliente-usuario; 7 - proyección del negocio; 8 - vocabulario del dominio; 9 - tecnología; 10 – involucrados/as; 11 – desarrolladores/as.

Figura 13 - Variables del contexto

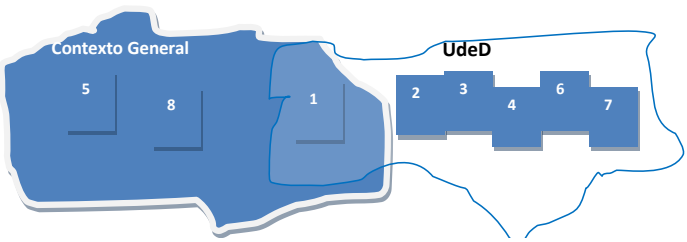
Aunque el contexto no esté bien definido, siempre se puede y se deben establecer los límites de la actuación el ingeniero de requisitos. Cuanto mejor este delineado el UdeD, mayores serán las oportunidades de construir un software completo y correcto. Pero cabe destacar que este UdeD se va precisando y mejorando sus límites durante toda la Ingeniería de Requisitos. Esto implica que algunas fronteras pueden estar algo confusas en las primeras etapas del proceso, pero deben quedar bien definidas al finalizar dicho proceso.

El UdeD no es estático. Va cambiando a medida que el tiempo transcurre. Estos cambios repercuten en el proceso del negocio y, por lo tanto, en los requisitos del software. Estas transformaciones que sufren los requisitos se denomina *evolución de los requisitos* y debe ser reflejado en los modelos construidos y gestionado apropiadamente.

Si bien el UdeD es el contexto particular para el desarrollo del nuevo sistema de software, existe otro contexto más general de la

organización, el cual puede influir en las decisiones que se deben tomar durante la construcción del nuevo software. También, existe el dominio de la aplicación que delimita el ámbito específico de negocio o sector para el cual se diseña la solución, o sea, proporciona el contexto y el conocimiento necesario para entender el problema y proponer soluciones apropiadas.

La Figura 14 muestra ambos contextos y un área de intersección que representa los aspectos del contexto general que influyen sobre el UdeD.



1. Costo de oportunidad del Sistema de Software; 2. Poco interés de los usuarios en el nuevo Sistema de Software; 3. Disponibilidad limitada de los cliente-usuario; 4. Imposibilidad para obtener información del dominio de la aplicación; 5. Falta de políticas organizacionales claras; 6. Acciones erráticas de los usuarios (ej., resolución de problemas que entran en conflicto con reglas del negocio o normativas); 7. Ausencia de procedimientos bien definidos; 8. Presiones de la competencia.

Figura 14 - Variables que afectan a la IR

A pesar de su importancia, las variables 1, 5 y 8 no suelen ser analizadas durante la Ingeniería de Requisitos, excluyendo temas que pueden ser decisivos para la construcción del software. Obviamente, la ubicación de las variables puede variar dependiendo de cada contexto particular debiendo ser analizadas en cada organización. Por ejemplo, el ítem 2 puede ser consecuencia directa del UdeD, pero también puede ser una consecuencia de la falta de compromiso de la dirección de la organización.

En la jerga de la Ingeniería de Requisitos los términos “contexto” y “universo de discurso” suelen ser utilizados como sinónimos.

6.2 Evolución del UdeD

Antes de describir la *evolución del UdeD* es necesario aclarar cuál es la acepción de la palabra "evolución" utilizada. Existe una controversia en la bibliografía, acerca del uso de este término. Para algunos autores el concepto está ceñido a las transformaciones que sufren los modelos durante todo el proceso de construcción del software, desde la Ingeniería de Requisitos, pasando por el diseño, el código y siguiendo con otras actividades hasta llegar a la puesta en servicio. Este es el caso de Brietman et. al [Brie05] donde se denomina evolución a las transformaciones de los escenarios y otros documentos durante todo el ciclo de vida del desarrollo de software. También existe una evolución relacionada con factores externos al sistema que generan cambios en su entorno, obligando a que este se adapte para seguir siendo relevante, funcional y alineado con los requisitos y las necesidades de las partes interesadas (cambios en: tecnología, marco legal, sociales, competencia, etc.). Otros, se refieren a la evolución como los cambios que sufre el contexto, producto de planificar la inserción de un nuevo sistema de software. En lo que sigue, se considera la acepción de la última definición.

La evolución del UdeD comienza con la mera decisión de incorporar un nuevo sistema de software, la cual habilita una nueva línea de pensamiento. Algo parecido sucede con el propio proceso de requisitos, el cual facilita la generación de nuevas ideas al tener que describir aspectos no observables o alterar, en diferente grado, el proceso del negocio existente. Otro aspecto que hace evolucionar el UdeD es que las organizaciones cambian; a veces por factores internos (tecnológicos,

políticas organizacionales, sociales, eventualmente culturales) y en otros casos por factores externos (legislación, competencia, etc.). Los cambios del UdeD, en algunos casos, son concurrentes y asincrónicos, esto implica que cuando uno comienza a desarrollarse el otro puede estar concluyendo. Cuando un proceso del negocio se ve afectado por diferentes cambios, se genera una inminente necesidad de evolucionar los requisitos de dicho software.

Puede observarse en la Figura 15 que existe un UdeD actual y otro UdeD futuro. Esta división en "actual" y "futuro" solo existe a efectos de mejorar la conceptualización del tema, ya que, como se mencionó, el UdeD es único. El *UdeD actual* describe el proceso del negocio como existe al momento de comenzar con el desarrollo del software y el *UdeD futuro* modela la planificación del proceso del negocio para cuando el sistema de software esté terminado. El gráfico de la Figura 15 muestra la evolución del contexto y está dividido en dos partes, A y B, con el objetivo de diferenciar la evolución real del UdeD de la "evolución" que se genera explícitamente dentro de la Ingeniería de Requisitos, lo que se ha denominado "evolución aparente".

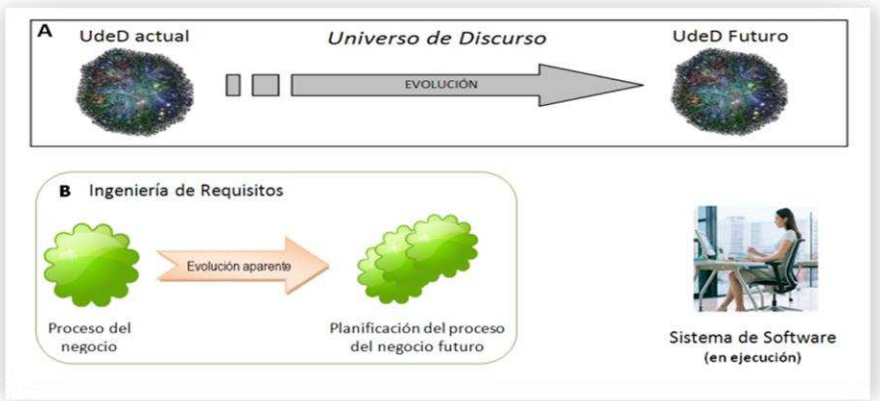


Figura 15 - Relación entre la evolución del UdeD y la IR

Parte de la evolución real se efectiviza un tiempo después de que el sistema de software se encuentra en ejecución, a diferencia de la evolución aparente que se genera con el objetivo de obtener un documento de especificación lo más completa posible. Esta evolución es conceptual ya que dichos cambios solo se pueden observar en los artefactos producidos en el mismo proceso de requisitos y de los cuales se desconoce que parte será aceptada en el UdeD y que parte se rechazará. Esta evolución se lleva a cabo en un lapso relativamente corto de tiempo.

La evolución del contexto genera la necesidad de construir procesos dinámicos que deben ser registrados correctamente para realizar el seguimiento desde cualquier punto del proceso de construcción del software hacia sus orígenes (backward traceability o trazabilidad hacia atrás) o desde sus orígenes hacia el código (forward traceability o trazabilidad hacia adelante).

6.2.1 Límites del sistema

Es sustancial remarcar que el UdeD no determina los límites del sistema. Es necesario ahondar en el UdeD, comprenderlo casi en su totalidad y conocer las necesidades del cliente-usuario para definirlo. Parafraseando a Jackson [Jack95], si bien la definición de los límites es lo primero que se debe hacer, es casualmente ese el momento en que menos se sabe al respecto. No solo se sabe menos, sino que no se tiene acceso a esa información, ya que la misma se irá completando y refinando durante toda la Ingeniería de Requisitos. Experiencias del mismo estudio replicado por diferentes personas, han demostrado que los límites que establecen esas distintas personas suelen ser diferentes. Esta diferencia puede representar solamente un innecesario exceso de

trabajo, pero también puede indicar haber ignorado aspectos importantes del problema.

6.3 Fuentes de información

Se denomina Fuente de Información (FI) a todo aquello que contiene, de diferente manera, la base de conocimiento de la organización y en particular del contexto específico que se desea estudiar. Las FI son parte del UdeD y se concentran en las *personas* y los *recursos*.
La *FI personas* son aquellas que tienen alguna influencia directa o indirecta sobre los requisitos del software. Como ya se mencionó, estas personas se denominan grupo *involucrados/as* y son la FI más importante. Este hecho se debe a que son poseedoras del mayor conocimiento del negocio y esto enriquece significativamente el proceso de requisitos, pero a su vez es la FI más compleja.

Cada involucrado ve el proceso del negocio desde una perspectiva propia o desde el cargo que desempeña en la organización. Una persona puede cambiar de punto de vista dependiendo del momento. A su vez, distintos involucrados/as pueden ver un proceso o tarea desde el mismo punto de vista o desde el contrario. Puede observarse en la

Figura 16 que la FI personas pueden tomar diferentes perspectivas o tener diferentes puntos de vista del contexto, dependiendo de sus interés, deseos, expectativas, miedos, etc.
Otra variante para analizar a los involucrados/as se basa en considerar su poder dentro de la organización y el dinamismo de cada persona dentro de ella. Estas variables pueden ser utilizadas para seleccionarlos.
Como se ve en la Tabla 4, existen cuatro grupos. Los del grupo A y B suelen ser fáciles de tratar; los del grupo C suelen tener pocas

expectativas y por lo tanto no suelen ser un problema; y los del grupo D son los de mayor atención ya que tienen poder y son poco predecibles.

Figura 16 - Puntos de vista y perspectivas de los involucrados/as

Poder dentro de la organización	Dinamismo	
	BAJO	**ALTO**
BAJO	**A** Pocos problemas	**B** Poco predecibles pero manejables
ALTO	**C** Poderoso pero predecibles	**D** Peligrosos u Oportunistas

Tabla 4 - Matriz de interés[8]

La *FI recursos* es todo aquello necesario para que las personas realicen sus tareas. Entre los recursos se encuentran:

- Toda la literatura del dominio (organigrama, políticas de la organización, manuales de usuario, manuales de procedimientos, normativas, contratos, formularios, libros,

[8] Tabla obtenida de Gardner et al. del libro "Handbook of Strategic Planning"

artículos en revistas especializadas, estándares nacionales e internacionales en uso o para aplicar, etc.).

- Sistemas de software propios y de terceros (COTS).
- Legislación vigente que afecta al dominio en estudio.

La importancia de las FI se determina, cuantitativa y cualitativamente, por la información intrínseca que contienen.

6.4 Información y conocimiento

Las FI contienen toda la información y el conocimiento indispensable para definir los requisitos del software. A continuación, la definición de cada una:

> **Rowley** (2007), *"La información es el resultado de organizar y procesar datos de manera que tengan un significado para quien los recibe".*
>
> **Nonaka & Takeuchi** (1995), *"El conocimiento es la capacidad de interpretar información y actuar basándose en ella, utilizando habilidades, experiencia y contexto"*.

La *información* es objetiva, puede ser transmitida de manera directa y sin interpretación personal. Suele representarse en tablas, gráficos, textos o bases de datos. Es universal, o sea que el mismo conjunto de información puede ser comprendido por diferentes personas si tienen el contexto adecuado. En general se utiliza para describir hechos por lo que no suele revestir problemas significativos.

El *conocimiento* es subjetivo, depende de la experiencia, habilidades y contexto de quien lo interpreta. Es de carácter dinámico, o sea que cambia y evoluciona con nuevas experiencias y aprendizajes. Es contextual, lo que se sabe o se comprende depende del entorno y la

situación. En general se utiliza para resolver problemas, tomar decisiones y realizar predicciones.

Si bien la Ingeniera de Requisitos debe trabajar con ambas, es más complejo el tratamiento del conocimiento que el de la información, principalmente cuando está en posesión de las personas. Esto se debe a que pueden tener varias perspectivas simultáneas, como ser el marco referencial de la organización, los objetivos, la cultura organizacional, etc. Además, este conocimiento puede estar explícito, implícito o tácito.

El *conocimiento explícito* [Nona95] [Pres15] es aquel que se encuentra en un nivel "consciente", es fácil de percibir ya que se utiliza para realizar las tareas. Se obtiene a través de la observación de las acciones de las personas, de las descripciones en la literatura del dominio, en los sistemas informáticos, en las entrevistas con los usuarios, en brainstorming, etc. Está claramente articulado, estructurado y documentado. Es fácil de comunicar y compartir porque está externalizado en forma de palabras, números, diagramas, manuales o bases de datos.

El *conocimiento implícito* [Gran96] [Spen96] [Tsou03] es el tipo de conocimiento que permanece en un nivel "inconsciente", se encuentra desarticulado y se utiliza de una manera mecánica sin que la fuente de información se dé cuenta de su contenido. Se refiere al conocimiento que está en posesión de una persona o grupo, pero que aún no ha sido formalizado ni comunicado de manera explícita. Este conocimiento suele ser difícil de detectar y se refiere a aspectos culturales de la organización, intereses propios de los usuarios, interpretaciones sobre la literatura del dominio, etc. Este tipo de conocimiento puede ser explicitado si se realiza el esfuerzo necesario.

El *conocimiento tácito* [Pola66] es profundamente personal y subjetivo, basado en habilidades, experiencias e intuiciones. Es difícil de formalizar o transmitir a otras personas, incluso con esfuerzo. Reside en la mente de las personas y es derivado de su experiencia, habilidades y contexto. Es difícil de articular o documentar porque está basado en intuiciones, prácticas y entendimientos implícitos.

Cross et al. [Cros01] identifican cuatro impulsores clave que afectan la transferencia de conocimiento:

- *Conciencia.* Reconocimiento de la existencia de conocimiento.
- *Acceso*. Facilidad para llegar al conocimiento y a las personas que lo poseen.
- *Compromiso.* Participación activa de los miembros en la creación y compartición de conocimiento.
- *Seguridad.* Confianza en que compartir conocimiento no tendrá consecuencias negativas.

Estos factores están interconectados y son esenciales para fomentar un entorno donde el conocimiento pueda fluir libremente entre los participantes. Construir este ambiente de trabajo es uno de los desafíos de la Ingeniera de Requisitos.

Referencias

[Bjør06] Bjørner, D. (2006). "Software Engineering 3, Domains, Requirements, and Software Design", Springer, ISBN-10 3-540-21151-9 Springer Berlin Heidelberg New York, pp 107.

[Brie05] Brietman K, Liete J., Berry D. (2005). "Supporting Software Evolution", Requirements Engineering, Volume 10, Issue 2, pp 112-131.

[Cros01] Cross, R., Parker, A., Prusak, L. y Borgatti, S. (2001). "Knowing What We Know: Supporting Knowledge Creation and

Sharing in Social Networks", Organizational Dynamics, Vol 30, N2, pp 100-120.

[Gran96] Grant, R. M. (1996). "Toward a knowledge-based theory of the firm". *Strategic Management Journal, 17*(S2), 109–122. https://doi.org/10.1002/smj.4250171110

[Jack95] Jackson, M. (1995). "Software Requirements & Specifications. A lexicon of practice, principles and prejudices", Addison-Wesley, Reading, MA/ACM Press.

[Nona95] Nonaka, I., & Takeuchi, H. (1995). *The Knowledge-Creating Company*. Oxford University Press.

[Pola66] Polanyi, M. (1966). "The Tacit Dimension". Routledge & Kegan Paul.

[Pres15] Pressman, R. S., & Maxim, B. R. (2015). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill Education.

[Spen96] Spender, J. C. (1996). "Making knowledge the basis of a dynamic theory of the firm". *Strategic Management Journal, 17*(S2), 45–62. https://doi.org/10.1002/smj.4250171106

[Tsou03] Tsoukas, H. (2003). "Do we really understand tacit knowledge?" *Knowledge Economy and Society Seminar*. London: LSE.

Capítulo

7 Requisitos del software

7.1 Requisito y requerimiento

Existe una controversia terminológica en el dominio de la Ingeniería de Requisitos relacionada con los términos *requisito* y *requerimiento*. Estos términos suelen ser utilizados como sinónimos, tanto en la literatura como en la jerga de los desarrolladores, pero no lo son. La etimología de estas palabras ya presentaba esta diferencia. Es así que en latín "petitio" tiene su traducción al castellano como *requerimiento* mientras que "requirens" como *requisito*. Las definiciones de la RAE - edición 22 se muestran en la Figura 17.

Requerimiento.

1. m. Acción y efecto de requerir.

Requerir.

1. tr. Intimar, avisar o hacer saber algo con autoridad pública.
2. tr. Reconocer o examinar el estado en que se halla algo.
3. tr. necesitar.
4. tr. Dicho de una persona: Solicitar, pretender, explicar su deseo o pasión amorosa.
5. tr. inducir (‖ instigar).

Requisito

m. Circunstancia o condición necesaria para algo.

Figura 17- Definición de requerimiento y requisito de RAE

Puede observarse que la definición de *requisito* contempla el concepto de "obligatoriedad", necesario cuando se debe especificar lo que el software "debe" hacer. Mientras que un *requerimiento* se refiere a las

solicitudes, necesidades y deseos del cliente-usuario que aún no han sido analizados desde su viabilidad, oportunidad, costo u otra característica del producto software. Por lo tanto, algunos requerimientos se convertirán en requisitos mientras que otros serán descartados.

Requirement

noun

1. that which is required; a thing demanded or obligatory: One of the requirements of the job is accuracy.
2. an act or instance of requiring.

Require

verb (used with object)

1. to have need of; need: He requires medical care.
2. to call on authoritatively; order or enjoin to do something: torequire an agent to account for money spent.
3. to ask for authoritatively or imperatively; demand.
4. to impose need or occasion for; make necessary orindispensable: The work required infini te patience.
5. to call for or exact as obligatory; ordain: The law requires annualincome-tax returns.

Requisite

adjective

1. required or necessary for a particular purpose, position, etc.;indispensable: the requisite skills of an engineer.

noun

2. something requisite; a necessary quality, thing, etc.

Request

noun

1. the act of asking for something to be given or done, especially as a favor or courtesy; solic itation orpetition: At his request, they left.
2. an instance of this: There have been many requests for the product.
3. a written statement of petition: If you need supplies, send in a request.
4. something asked for: to obtain one's request.
5. the state of being asked for; demand.

Figura 18- Definición de requerimiento y requisito en inglés[9]

El uso incorrecto de la palabra *requerimiento*, puede ser originado en lo que se conoce como "falsos amigos" entre idiomas, que consiste en tener palabras que a pesar de tener un significado diferente se las escribe o pronuncian de una manera similar. Este caso de "amigos falsos" se presenta en la transliteración del término en inglés "requirements" al término en castellano *requerimientos*. Puede

[9] http://dictionary.reference.com

observarse en la Figura 18 que el término "requirement" se corresponde con la definición previa de *requisito.*

7.2 Requisitos del software

Los requisitos expresan los servicios y las restricciones impuestas a un sistema de software para dar solución a un problema o necesidad del mundo real [Jack95] [Koto98] [Swe04]. Algunas definiciones:

> **IEEE 610-1990**, "*1. Una condición o capacidad que necesita un usuario para resolver un problema o alcanzar un objetivo.*
>
> *2. Una condición o capacidad que debe poseer un sistema o componente de un sistema para satisfacer un contrato, un estándar, una especificación, u otro documento formalmente impuesto.*
>
> *3. Una representación documentada de una condición o una capacidad como se describe en 1 y 2".*
>
> **Young** (2004), "*un atributo necesario en un sistema, una sentencia que identifica una capacidad, características o un factor de calidad de un sistema en función de tener valor y utilidad a un cliente o usuario*".

Los requisitos deben estar bien definidos, ser correctamente comprendidos por todos los que participan en la construcción del software y asegurar que el producto final cumpla con las expectativas y necesidades del cliente. Cuando se está en presencia de requisitos pobres o de baja calidad, es altamente probable que el proyecto sufra costos adicionales, retrasos, pérdida de confianza del cliente, entre otros problemas. Obtener requisitos del software de la mejor calidad posible es responsabilidad de la Ingeniería de Requisitos.

7.3 ¿Los requisitos se construyen o se elicitan?

Algunas visiones sostienen que los requisitos ya existen en el contexto. Otras aluden que los requisitos deben ser construidos. Sin lugar a dudas no se puede estar en una única postura ya que algunos requisitos son preexistentes y otros deben ser construidos.

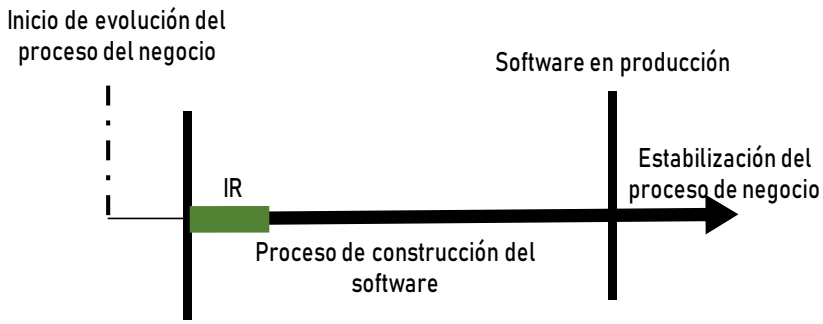


Figura 19 – La IR en el proceso de construcción del software

En la Figura 19 se puede observar que los cambios en el proceso del negocio se inician antes de comenzar la construcción del software. Esto significa que la organización, a partir de la decisión de incluir un sistema informático para satisfacer una necesidad, ya está pensando en la idea. De esta manera, cuando inicia el proceso de requisitos ya existen deseos, expectativas, requerimientos y restricciones provenientes de los involucrados/as. Cada propuesta será analizada para decidir si se convierte efectivamente en un requisito del software o si es descartada por ser innecesaria, no viable o entrar en conflicto con otros requisitos.

Retomando la pregunta inicial, se puede concluir que existen algunos requisitos que se elicitan del contexto mientras que otros deberán ser construidos.

7.4 Quién define los requisitos

En el dominio de los sistemas de información se suele pensar que los requisitos son definidos por el *cliente-usuario*. Es posible que esto ocurra, pero probablemente estas personas no tengan la información suficiente como para concebir el conjunto de prestaciones posibles del sistema de software. Otros piensan que los requisitos deben ser definidos por los desarrolladores/as del sistema. Esto puede ocurrir, pero probablemente estas personas no tengan la información suficiente acerca del proceso del negocio y menos aún del impacto que el sistema de software tendrá en la organización. Parece ser que la mejor solución es que los requisitos sean **construidos** de manera colaborativa. En síntesis, algunos requisitos los define el cliente (por ejemplo, cuando ya existen en el contexto o son aspectos esenciales para el negocio), otros los define el ingeniero/a de requisitos (por ejemplo, para asegurar la consistencia o seguridad del software) y habrá otros requisitos que serán construidos en forma colaborativa (por ejemplo, al definir alternativas posibles o durante la negociación). Para lograr esto, es necesario que la actitud que debe adoptar el ingeniero/a de requisitos no sea preguntar qué es lo que quiere el cliente, porque de esa manera incentiva la definición de los requisitos por parte del mismo, sino preguntar "***¿qué hace?***" para promover que el ingeniero/a de requisitos busque información sobre las características del proceso del negocio y esto lo habilite a participar en la definición de los requisitos. Aquí es muy importante comprender que hubo un acercamiento entre el cliente con el equipo de desarrollo, porque fue el cliente quien manifestó la necesidad de incorporar un sistema de software para un objetivo específico, siendo esto central para el proceso de requisitos ya que define la intención del cliente, lo que se conoce como *objetivo general*

del sistema. Lo próximo que se debe hacer es planificar el proceso del negocio futuro, suponiendo la existencia del software a construir. Esta actividad la debe desarrollar en conjunto el cliente-usuario y el ingeniero de requisitos. Una vez hecho esto, los requisitos habrán quedado definidos en un documento o en un modelo. Naturalmente que el peso de construcción de los modelos recae sobre el ingeniero de requisitos y la participación del cliente-usuario se materializa en entrevistas, seleccionando alternativas, validando, negociando, etc. Para que el cliente-usuario pueda cumplir acabadamente su rol, es necesario que comprenda los modelos que se construyen. En este sentido los modelos se transforman en el medio más importante de comunicación. Es necesario enfatizar esto, tanto como sea posible, ya que, al seleccionar opciones, al responder consultas, o al validar el proceso del negocio planeado, el cliente-usuario toma decisiones importantes que afectarán en forma directa a la organización. Es responsabilidad del ingeniero de requisitos una descripción clara y precisa de los requisitos y es responsabilidad del cliente-usuario asegurarse que comprende suficientemente bien los modelos como para analizar la viabilidad del software en función de los objetivos establecidos.

7.5 Propiedades de los requisitos

Una *propiedad* es una característica esencial que garantiza la calidad de cada requisito. De esta manera, se asegura un alineamiento con las necesidades del cliente y con el *objetivo del sistema*. A continuación, se describen algunas *propiedades* de los requisitos:

- *Correcto*. Refleja lo que realmente se necesita.
- *Factible***.** Realizable dentro de las limitaciones del proyecto.

- *Necesario***.** Imprescindible para el sistema. Si se elimina existirá una deficiencia que no puede ser suplida por otras capacidades del sistema.
- *Libre de implementación***.** Evita imponer restricciones innecesarias al diseño arquitectónico.
- *Verificable***.** Se puede probar si ha sido cumplido por el propio sistema.
- *Completo***.** No necesita mayor ampliación porque es mensurable y suficientemente descriptivo.
- *Consistente***.** No se contradice ni entra en conflicto con otros requisitos.
- *No ambiguo***.** Es claro y preciso. Solo puede ser interpretado de una manera.
- *Trazable***.** Puede ser seguido hacia los documentos que le dieron original, a los requisitos de nivel superior u otra fuente o hacia los requisitos específicos.
- *Modificable***.** Es fácil de actualizar sin causar confusión.
- *Singular***.** Incluye solo un requisito sin uso de conjunciones.

Las propiedades aseguran la calidad de cada requisito y cuando alguna de ellas falla, todas entran rápidamente en conflicto o tensión. Por ejemplo, un requisito ambiguo no permite asegurar ninguna de las otras características.

Para ejemplificar el uso de las propiedades se analizan cuatro requisitos tomados al azar de la ERS del Sistema Gestión de ATM[10] (ver Anexo) y se analizan algunas de sus propiedades.

[10] **ATM=** Automated Teller Machine (en español Cajero Automático)

Requisito #1: *El sistema debe asignar un número identificatorio secuencial a cada ATM.*
Dudas del ingeniero de requisitos: -
Cumplimiento de las características del requisito:

Correcto	√	Factible	√	Necesario	√
Verificable	√	Completo	√	Consistente	√
No ambiguo	√	Rastreable	√	Modificable	√

Comentario final: Este requisito puede ser incorporado al documento de especificación.

Requisito #2*: El sistema debe imprimir etiquetas con los números de serie de los ATM de manera unitaria*.
Dudas del ingeniero de requisitos: ¿con "unitaria" se refiere a un solo ATM o a una única etiqueta?
Cumplimiento de las características del requisito:

Correcto	√	Factible	√	Necesario	√
Verificable	X	Completo	X	Consistente	X
No ambiguo	X	Rastreable	X	Modificable	X

Comentario final: Este requisito debe ser revisado, completado y vuelto a verificar.

Requisito #3: *El sistema debe generar reportes de las configuraciones de los ATM de manera semanal, mensual y anual, según sea la exigencia*.
Dudas del ingeniero de requisitos: ¿Cómo se determina la exigencia?
Cumplimiento de las características del requisito:

Correcto	√	Factible	√	Necesario	X
Verificable	X	Completo	X	Consistente	√
No ambiguo	X	Rastreable	√	Modificable	√

Comentario final: Este requisito debe reescrito sin expresiones vagas y vuelto a verificar.

Requisito #4: *El sistema debe registrar los pedidos aun sin tener todos los datos obligatorios completos.*
Dudas del ingeniero de requisitos: ¿Cómo se sabe a quién corresponde el pedido o cuál es la fecha del pedido para calcular la entrega? …
Cumplimiento de las características del requisito:

Correcto	X	Factible	X	Necesario	X
Verificable	X	Completo	X	Consistente	X
No ambiguo	X	Rastreable	X	Modificable	X

Comentario final: Este requisito debe ser eliminado o reformulado.

Se puede observar que el Requisito #1 cumple con todas las características y de esta manera, se garantiza la calidad individual del mismo. En el Requisito #2 falla la completitud, esto significa que le falta información. Inmediatamente pone en tensión a las demás propiedades ya que deja de ser verificable, consistente, rastreable, etc.

Cuando un requisito es *correcto*, pero no es *necesario*, como es el caso del Requisito #3, posiblemente pueda ser incorporado en una versión posterior del software, pero debería ser eliminado de esta especificación para darle prioridad a los necesarios. En el Requisito #4 aparece un requisito que no es correcto y esto anula inmediatamente al resto de las propiedades. Este requisito debe ser enviado a revisión para determinar si debe ser eliminado o está mal planteado. Todos los requisitos que no cumplen con alguna propiedad deben ser revisados y corregidos antes de ser incorporados a la especificación de requisitos para garantizar la calidad de la misma.

Además de pensar en cada requisito de manera particular, se debe considerar el conjunto de requisitos de la ERS. Este conjunto debe ser *completo*, deben estar todos los requisitos que satisfacen el objetivo del sistema*; consistente*, sin contradicciones ni duplicados; *asequible*, puede ser realizado dentro de las limitaciones del proyecto como el cronograma, leyes, presupuesto y, finalmente, *acotado*, debe cumplir específicamente con el objetivo del sistema.

7.6 Atributos de los requisitos

Un atributo es un metadato o información adicional asociada a cada requisito que permite comprenderlo y gestionarlo a lo largo de todo el proceso de construcción del software. Algunas definiciones:

> **Kotonya & Sommerville** (1998), propone un meta-modelo para especificar requisitos con los siguientes datos: *"el identificador del requisito, descripción del requisito, fecha de creación, fecha de modificación, origen del requisito (persona, documento u otro requisito), fundamento, estado, lista de requisitos dependientes, lista de requisitos de los que depende, vínculos a modelos y un*

comentario".

Davis (1999), propone para la gestión de requisitos los siguientes atributos: *"beneficio del cliente, esfuerzo de implementación, prioridad de desarrollo, estado, autores, parte responsable, fundamento, fecha de creación o modificación, versión y relación con otros requisitos. Donde autores corresponde al responsable de escribir la ERS o quién identificó la necesidad; y la parte responsable es quien asegura la satisfacción del requisito"*.

SWEBOK (2014), propone tanto para la gestión como para la interpretación de los requisitos, incluir los siguientes atributos: *"la clasificación en distintas dimensiones (RF o RNF, derivado o impuesto, sobre el producto o el proceso, prioridad, alcance, volatilidad - estabilidad), el método de verificación o plan de prueba de aceptación, el fundamento, el origen del requisito, la historia de cambios y un identificador único para cada requisito"*.

A continuación, se describen algunos atributos:

- *Identificador único.* Código único para identificar cada requisito.
- *Descripción.* Definición clara y precisa del requisito.
- *Fundamento.* Razón por la cual el requisito es necesario.
- *Origen.* Origen del requisito (cliente, usuario, normativa).
- *Estado.* Situación actual del requisito en el ciclo de vida del proyecto.
- *Prioridad.* Nivel de importancia o urgencia del requisito.
- *Estabilidad.* Probabilidad de que el requisito cambie en el futuro.
- *Dependencia.* Se debe definir la dependencia entre requisitos, cuando ésta exista.

- *Asignación.* Quién es responsable de implementar/verificar el requisito.
- *Riesgo.* Nivel de riesgo asociado a la implementación del requisito.
- *Criterios de aceptación.* Condiciones que deben cumplirse para aceptar el requisito.
- *Versión.* Número de versión del requisito.
- *Complejidad.* Nivel de dificultad técnica del requisito.
- *Fecha de creación.* Cuándo fue documentado el requisito.
- *Fecha límite.* Plazo de entrega o cumplimiento del requisito.
- *Trazabilidad.* Capacidad de rastrear el requisito hacia su origen o destino.
- *Dificultad.* Se debe anotar la dificultad asumida para cada requisito.

La determinación de qué atributos considerar en un proyecto depende de la tolerancia a fallos o la falta de cumplimiento que se pueda aceptar para que no afecte la integridad del software. Como sucede con las propiedades, determinar qué atributos asociar a los requisitos es una decisión del ingeniero de requisitos.

7.7 Clasificación de los requisitos

Sommerville clasifica los requisitos en *Requisitos de Usuario* y *Requisitos del Sistema*. Los de usuario son declaraciones de los servicios que espera el usuario y de las restricciones que considera necesarias para el nuevo software. Los del sistema son descripciones más detalladas de las funciones, los servicios y las restricciones operacionales del sistema de software. También habla de *Requisitos del Dominio* que son aquellos que se derivan del dominio de la aplicación

del sistema, más que a partir de las necesidades específicas de los usuarios del sistema.

Young define cuatro tipos de requisitos: *Requisitos del negocio*, *Requisitos del usuario*, *Requisitos del producto* y *Requisitos del ambiente*. Los requisitos del negocio son la base del desarrollo del software y son derivados de los objetivos del negocio. Mientras que los del usuario capturan la visión del cliente, el alcance del sistema, la estimación de costos, etc. necesarios para construir el sistema de software. Los requisitos del producto son los del desarrollo. Los del ambiente son los que se determinan por aspectos sociales, físicos y culturales en el cual el sistema de software será utilizado.

Para Wiegers et al. [Wieg13] los requisitos se clasifican por algún criterio funcional, por su origen, por algún aspecto temporal o por algún otro criterio que se adapte mejor al contexto y propone separar en *Requisitos del Negocio*, *Requisitos del Usuario* y *Requisitos del Software*. Los del negocio describen los objetivos estratégicos o las metas de alto nivel que una organización desea alcanzar con el sistema. Están relacionados con el *por qué* se está desarrollando el software, definiendo las razones comerciales o el valor que se espera lograr. Los del usuario describen lo que necesitan del sistema para cumplir con sus tareas y objetivos. Están enfocados en las interacciones y la experiencia del usuario con el sistema. Suelen documentarse en forma de historias de usuario**,** casos de uso o escenarios y, los del software definen las funcionalidades y características técnicas que el sistema debe proporcionar para satisfacer los requisitos del usuario y del negocio.

Dentro de las clasificaciones recién mencionada, existe otra que es transversal y se refiere a los aspectos *funcionales* y *no funcionales.*

Los *Requisitos funcionales* (RF) son conocidos y aceptados como capacidades o servicios que el sistema de software deberá satisfacer. A continuación, algunas definiciones:

> **Sommerville** (2011), "*Son enunciados acerca de servicios que el sistema debe proveer, de cómo debería reaccionar el sistema a entradas particulares y de cómo debería comportarse el sistema en situaciones específicas. En algunos casos los requerimientos funcionales también explican lo que no debe hacer el sistema".*

Los *Requisitos no funcionales* (RNF) se los identifica por ciertas cualidades-atributos-propiedades del nuevo sistema de software, como ser restricciones de la solución y algunos aspectos de calidad [Cysn04] [Ulla11] [Pres15] [Lapl17]. Los RNF serán operacionalizados al convertirlos en criterios verificables y acciones concretas que permitan su diseño, implementación y evaluación. Cabe remarcar que algunos RNF serán operacionalizados durante la Ingeniería de Requisitos mientras que otros se convertirán durante el diseño. Es responsabilidad de la Ingeniería de Requisitos operacionalizar aquellos RNF que afecten el funcionamiento del software o altere el proceso de negocio futuro donde operará el nuevo sistema de software. A continuación, algunas definiciones:

> **Kotonya & Sommerville** (1998), "*Mientras que los requisitos funcionales abordan* qué *hace un sistema, los requisitos no funcionales se enfocan en* cómo *el sistema lo hace*".
>
> **Jacobson & Booch & Rumbaugh** (1999), "*Es un requerimiento que especifica propiedades del sistema, tales*

como restricciones de implementación y de ambiente, performance, dependencias de plataforma, mantenimiento, capacidad de extensión y confiabilidad. Un requerimiento que especifica restricciones físicas sobre un requerimiento funcional".

Cysneiros & Liete (2002), *"Son requerimientos de calidad, que representan restricciones o las cualidades que el sistema debe tener tales como: precisión, usabilidad, seguridad, rendimiento, confiabilidad, performance, entre otras".*

Laplante (2017), *"A diferencia de los requisitos funcionales, que describen qué debe hacer el sistema, los requisitos no funcionales se centran en las restricciones y propiedades del sistema, como su tiempo de respuesta o su capacidad de manejo de fallos".*

En la Tabla 5, se ejemplifica cada tipo de requisito basado en la clasificación de Wiegers (*Negocio*, *Usuario* y *Software).*

Tipo de Requisito	Requisito Funcional	Requisito No Funcional
Negocio	El sistema debe permitir al cliente realizar compras en línea para aumentar las ventas en un 20% en un año.	El sistema debe estar disponible 24/7 para asegurar la accesibilidad global y maximizar las ventas.
Usuario	El usuario debe poder consultar su historial de cargas de configuraciones realizadas en las últimas 48 hs.	El sistema debe ser accesible desde una computadora y dispositivos móviles, compatible con diferentes navegadores y sistemas operativos.

Tipo de Requisito	Requisito Funcional	Requisito No Funcional
Software	El sistema debe determinar cuál es la hoja de Excel con el configurador adecuado para cada ATM.	El sistema debe requerir la autenticación de los usuarios autorizados para cargar la configuración en los ATM.

Tabla 5 - Ejemplos de Tipos de Requisitos

7.8 Recomendaciones para mejorar la escritura de los requisitos

Los requisitos deben indicar "qué" se necesita y no "cómo" lo hace [Wieg99]. Los requisitos deben tratar de no incluir decisiones de diseño. Estas decisiones serán realizadas en el momento oportuno permitiendo que sean las óptimas en cada caso, sin limitarlas innecesariamente.

Para describir los requisitos se deben evitar términos vagos y generales que resultan en requisitos que a menudo son difíciles o incluso imposible de verificar o pueden generar múltiples interpretaciones. Según la ISO/IEC/IEEE 29148:2018[11] se debe evitar:

- Superlativos (como "mejor", "más").
- Lenguaje subjetivo (como "fácil de usar", "fácil de usar", "rentable").
- Pronombres vagos (como 'eso', 'esto', 'eso').

[11] Estándar internacional "*Systems and software engineering - Life cycle processes - Requirements engineering"* que establece las mejores prácticas y directrices para la Ingeniería de Requisitos, abarcando la recolección, análisis, especificación, validación y gestión de requisitos.

- Adverbios y adjetivos ambiguos (como "casi siempre", "significativo", "mínimo").
- Términos abiertos y no verificables (como "brindar soporte", "pero no limitarse a", "como mínimo").
- Frases comparativas (como "mejor que", "mayor calidad").
- Lagunas (como "si es posible", "según corresponda", "según corresponda").
- Referencias incompletas (sin especificar la referencia con su fecha y número de versión; sin especificar solo las partes aplicables de la referencia para restringir el trabajo de verificación).
- Declaraciones negativas (como declaraciones sobre la capacidad del sistema que no se deben proporcionar).

La norma establece que la mejor manera de expresar un requisito es utilizando *sentencias declarativas*, ya que describen qué debe hacer el sistema o qué condiciones debe cumplir, sin detallar cómo se debe lograr ese comportamiento. Este enfoque fomenta la flexibilidad y claridad en la definición de los requisitos, y facilita la verificación y validación del sistema final. Por ejemplo, en vez del requisito "El sistema debe permitir a los usuarios autenticarse mediante credenciales válidas" se debe utilizar el requisito "El sistema debe implementar un formulario de inicio de sesión con tres campos para nombre de usuario, contraseña y código CAPTCHA[12]". Sin embargo, es importante que los requisitos mantengan un equilibrio entre especificidad y flexibilidad. Si los campos a validar en el inicio de sesión aún están sujetos a cambios o depende de decisiones futuras, el siguiente requisito podría ser el

[12] **Completely A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part: test de Turing público y automático para distinguir a los ordenadores de los humanos.

adecuado "El sistema debe implementar un formulario de inicio de sesión que incluya campos para nombre de usuario, contraseña y un mecanismo de validación adicional, como un código CAPTCHA". De esta manera se permite que el sistema sea verificable y específico, pero con cierta flexibilidad para adaptarse a posibles cambios.

Referencias

[Cysn02] Cysneiros L.M., Leite J.C. (2002). "Non-functional requirements: from elicitation to modelling languages", Proceedings of the 24th international conference on Software engineering, pp. 699-700.

[Cysn04] Cysneiros L.M., Yu. E (2004). "Non-functional requirements elicitation", En el libro de Prado Leite, J.C.S., Doorn, J.H. (eds) Perspectives on Software Requirements. Kluwer Academic Publishers, EEUU, ISBN: 1-4020-7625-8, Capítulo 6.

[ISO29148] "Systems and software engineering — Life cycle processes — Requirements engineering". (2018). ISO/IEC/IEEE 29148. https://normasiso.org/norma-iso-29148/#google_vignette

[IEEE610] IEEE Std 610-1990. "IEEE Standard Glossary of Software Engineering Terminology", IEEE.

[Jack95] Jackson, M. (1995). "Software Requirements & Specifications. A lexicon of practice, principles and prejudices", Addison-Wesley, Reading, MA/ACM Press

[Koto98] Kotonya G. y Sommerville I. (1998). "Requirements Engineering: Processes and Techniques", John Wiley & Sons.

[Lapl17] Laplante, P. A. (2017). "Requirements engineering for software and systems" (3rd ed.). CRC Press.

[Pres15] Pressman, R. S., & Maxim, B. R. (2015). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill Education.

[Somm11] Sommerville I. (2011). "Ingeniería de Software", Pearson, Edition 9.

[SWEB14] "Software Engineering Body of Knowledge (SWEBOK)", Institute of Electrical and Electronics Engineers (IEEE), Cap 2 pp.1, pp.2-2, 2014.

https://www.computer.org/education/bodies-of-knowledge/software-engineering

[Ulla11] Ullah, S., Iqbal, M., & Khan, A. M. (2011). "A survey on issues in non-functional requirements elicitation". In International Conference on Computer Networks and Information Technology (pp. 333-340). IEEE.

[Wieg99] Wiegers, K.E. (1999). "Writing Quality Requirements - Process Impact". Disponible en: http://www.processimpact.com/articles/qualreqs.pdf

[Wieg13] Wiegers, K., & Beatty, J. (2013). "Software requirements", (3rd ed.). Microsoft Press.

[Youn04] Ralph Young (2004). "The Requirements Engineering Handbook", ISBN: 1-58053-266-7, pp.1.2, pp.49-55, Artech House.

Capítulo

8 Modelos que contienen requisitos

8.1 Uso del lenguaje natural

Durante muchos años la frase “una imagen vale más que mil palabras” resultó casi un dogma en la construcción de software. Esta idea se complementaba con el pensamiento acerca de que el lenguaje corriente generaba “vaguedad y embrollo” en las representaciones [Gane77]. De esta manera se justificó el uso excesivo de modelos gráficos que el cliente-usuario no podía comprender. Esta idea se fue desvaneciendo al vislumbrar que muchos proyectos fracasaban debido a especificaciones de requisitos que no representaban la necesidad del cliente. Para contrarrestar este problema la comunidad de Ingeniería de Requisitos buscó un denominador común entre el grupo desarrolladores/as y el grupo involucrados/as que asegure una asertiva participación de todos en el proceso. El resultado fue el uso del lenguaje natural (LN) ya que todos podían comprenderlo con un ínfimo esfuerzo de aprendizaje. Esta pequeña dificultad del LN se debe a la necesidad de estructurarlo mínimamente para evitar algunos problemas endógenos, como la ambigüedad y la imprecisión [Gerv98] [Berr08] [Gerv05] del mismo. Estos problemas pueden ser controlados con algunas formalidades sin que afecte la legibilidad y simplicidad del texto, como por ejemplo el

uso de glosarios [Ryan15]; la verificación de los modelos [Faga76] [Leit05] [Bexy06]; la generación y análisis de modelos previos a la especificación de requisitos (historias de usuario, casos de uso, escenarios); entre otros.

A continuación, se describen algunas recomendaciones de Berry [Berry04] para utilizar de manera efectiva el lenguaje natural. Estas recomendaciones parecen triviales, pero no lo son, ya que esconden un cierto grado de entrenamiento de los ingenieros/as de requisitos y la creación de técnicas que permitan analizar las descripciones generadas:

- Aprender a escribir con menos ambigüedad e imprecisión.
- Aprender a detectar la ambigüedad y la imprecisión.
- Usar un lenguaje natural restringido para que sea menos ambiguo y más preciso.

En el uso del lenguaje natural, la Ingeniería de Requisitos ha propuesto diferentes modelos, siendo los más conocidos:

- *Historias de Usuario* (User Story)
- *Casos de Uso* (Use Case)
- *Escenarios* (Scenarios)

Las *historias de usuario* se concentran en las necesidades y expectativas del usuario; los *casos de uso* están centrados en la interacción de los actores con el sistema y los *escenarios* abordan el contexto real y la experiencia del usuario. Como puede observarse cada modelo mencionado tiene una perspectiva particular. Es importante remarcar que estos modelos no son intercambiables, sino que pueden ser utilizados de manera complementaria o conjunta, por ejemplo, utilizar las historias de usuario y luego, documentar las más relevantes con casos de uso o escenarios ya que estos tienen un nivel de detalle superior. La elección de qué modelo utilizar en cada proyecto es una

decisión del ingeniero/a de requisitos y dependerá del contexto, del enfoque del proyecto y del nivel de detalle requerido.

8.2 Historias de Usuario

Una *historia de usuario* (HU) [Cohn04] [Patt14] es una descripción breve de una funcionalidad o característica del sistema desde la perspectiva del usuario final. Estas funcionalidades o requisitos son flexibles, adaptables y colaborativos. Las historias de usuario son las más utilizadas en las metodologías ágiles como Scrum o Kanban. No son adecuadas cuando se necesita un alto nivel de detalle o una descripción exhaustiva de cómo debe funcionar el sistema. Su objetivo es expresar de manera simple y directa lo que un usuario espera del producto software, enfocándose en los beneficios que obtendrá.

Figura 20 - Historias de Usuario

Se puede observar en la Figura 20 que las historias de usuario se pueden gestionar en un *panel* o *tablero* que es una herramienta visual utilizada en metodologías ágiles para gestionar y rastrear el progreso de las historias de usuario en el ciclo de desarrollo. Este tablero ayuda al equipo a mantenerse organizado, priorizar tareas, y tener visibilidad del

estado de cada historia en tiempo real. El panel no solo facilita la gestión de historias de usuario, sino que también fomenta la transparencia y colaboración entre los miembros del equipo.

Una historia de usuario se define de la siguiente manera:

- **Como [actor]**: describe quién es el usuario que necesita la funcionalidad (puede ser un cliente, administrador, etc.).
- **Quiero [acción o funcionalidad]**: describe qué quiere hacer o lograr el usuario.
- **Para [beneficio]**: explica el valor o la razón detrás de la solicitud. ¿Por qué es importante?

Cada historia de usuario debe estar acompañada de los *criterios de aceptación*, que son condiciones específicas que deben cumplirse para que la historia sea considerada completa.

Como responsable de compras (usuario),
Quiero reservar números de OC (Orden de Compra),
Para asegurar que ciertos clientes especiales tengan un número de OC reservado para próximas compras.

CRITERIOS DE ACEPTACION:

- El usuario debe poder acceder a la funcionalidad de reservar números de OC en el sistema solo si está autorizado en el sistema.
- El sistema debe calcular un rango de números de OC donde el mínimo es el "último número de OC utilizado +1" y 10 números posteriores disponibles (Ej. ultimo número de OC = 14, el sistema muestra de 15 a 25 siempre que en ese rango no existan números previamente reservados).
- El sistema debe permitir seleccionar los números de OC a reservar.
- El sistema debe controlar que no se superen los números de OC permitidos por mes.
- El usuario debe poder ver la lista de los números de OC reservados.
- El sistema debe registrar el responsable, fecha, hora y números de OC reservados.
- El sistema debe avisar con un mensaje de éxito que fueron reservados correctamente.
- El sistema debe permitir liberar números de OC reservados que no sean utilizados en un plazo de 15 días.

Figura 21 - HU y criterios de aceptación

Los criterios de aceptación detallan lo que debe cumplir la historia de usuario y deben ser verificados para asegurar que la funcionalidad se implementó correctamente.

Ron Jeffries (cofundador de XP) determinó que existen tres "C" que deben ser tenidas en cuenta para gestionar correctamente las historias de usuario y que sintetizan su implementación:

- **C**ard (Tarjeta). Construir cada historia de manera clara y breve.
- **C**onversation (Conversación). Generar un diálogo constante entre los desarrolladores/as y los involucrados/as con el objetivo de ir refinando la historia.
- **C**onfirmation (Confirmación). Crear los criterios de aceptación que permitan verificar que la historia cumple con las expectativas.

En el capítulo 14 se pueden encontrar otros detalles relacionados con las metodologías ágiles y el uso de las historias de usuario.

8.3 Casos de uso

Un *caso de uso* (CU) [Jaco92] [Dano97] [Booc99] es una descripción detallada de cómo el sistema interactúa con usuarios u otros sistemas externos para lograr un objetivo específico. Es una representación adecuada para sistemas complejos y cuando se necesita capturar flujos alternativos y excepciones. Ayudan cuando:

- *Se requiere requisitos detallados.* Proporcionan una descripción estructurada y detallada de cómo interactúan los actores con el sistema para cumplir un objetivo específico.
- *Existen muchas interacciones.* Ayudan a visualizar cómo se comporta el sistema desde la mirada de diferentes actores.
- *Se está en una transición entre el análisis y el diseño.* Describen el comportamiento del sistema de manera técnica pero comprensible para los desarrolladores/as y otros miembros del equipo técnico.

Los casos de uso suelen representarse de forma textual, aunque también pueden complementarse con *diagramas de casos de uso* en UML (Unified Modeling Language) [Booc99] [Daou12]. En general, los casos de uso no son utilizados en proyectos pequeños o cuando no es necesario el nivel de detalle que proporcionan, ya que el esfuerzo de generarlos y mantenerlos actualizados puede ser significativo.

Jacobson introdujo los casos de uso como una técnica para capturar los requisitos funcionales de un sistema, clasificándolos según su nivel de abstracción y detalle:

- *Casos de uso de nivel de resumen* (Summary-level Use Cases). Son casos de uso de alto nivel que describen la funcionalidad principal del sistema, sin entrar en detalles sobre su implementación o pasos específicos. Se centran en el *porqué* del sistema y en las metas del usuario final. Sirven para comunicar una visión general a los involucrados/as y conectar con los objetivos de negocio.
- *Casos de uso de nivel de usuario* (User-level Use Cases). Detallan las interacciones concretas entre el usuario y el sistema desde el punto de vista funcional. Se vinculan a tareas específicas que el usuario realiza. Se centran en el *qué* quiere lograr el usuario con el sistema, sin especificar cómo lo logra técnicamente. Ayudan a capturar los requisitos funcionales específicos que el sistema debe cumplir.
- *Casos de uso de nivel de sub-función* (Subfunction-level Use Cases). Describen funciones internas del sistema necesarias para soportar los casos de uso de nivel de usuario. Son más detallados y técnicos. Se centran en el *cómo* el sistema implementará ciertas funciones técnicas. Son valiosos para

diseñadores/as y desarrolladores/as, ya que guían la implementación técnica.

Grady Booch, coautor del lenguaje UML junto con Jacobson y Rumbaugh [Jaco99], complementa el enfoque de Jacobson al considerar los casos de uso como elementos dentro de un proceso de modelado más amplio. Booch no clasifica explícitamente los casos de uso en tipos, pero su contribución se enfoca en que:

- Sugiere que los casos de uso deben ser la base para derivar la arquitectura del sistema. Identificar *escenarios clave* para definir los componentes y subsistemas. Usar los casos de uso para modelar tanto los *aspectos estáticos* (como clases) como los *aspectos dinámicos* (como interacciones y comportamiento).
- Propone estructurar los casos de uso en niveles jerárquicos, similares a la propuesta de Jacobson, pero enfatiza el uso de diagramas UML para representarlos gráficamente. Esto incluye:
    - *Casos de uso principales.* Similar a los casos de uso de nivel de resumen de Jacobson.
    - *Casos de uso secundarios.* Se alinean con los de nivel de usuario.
    - *Casos de uso extendidos y de inclusión.* Relacionados a través de relaciones UML como *include* o *extend*.
- Destaca la importancia de descomponer los casos de uso grandes en componentes modulares para que puedan ser implementados de manera independiente, promoviendo la reutilización de componentes.

Jacobson ofrece una visión práctica con un enfoque *top-down* para capturar los requisitos funcionales a través de casos de uso ya que identifica objetivos globales y los descompone en interacciones

específicas. Booch, que también utiliza un enfoque *top-down,* se centra en cómo estos casos de uso guían el diseño arquitectónico y técnico del sistema. Rumbaugh presenta un enfoque híbrido de construcción de los casos de uso al combinar el enfoque *top-down* cuando realiza un análisis de alto nivel y un enfoque bottom-*up* cuando analiza las funcionalidades concretas. Beck en XP utiliza un enfoque *bottom-up* ya que comienza desde historias pequeñas e iterativas que responden a necesidades inmediatas.

Existen múltiples formas de representar un caso de uso [Cock00] pero se pueden mencionar algunos componentes comunes:

- *Actor.* Personas, sistemas o dispositivos externos que interactúan con el sistema. Pueden ser actores primarios (quienes inician la acción) o secundarios (quienes ayudan en el proceso).
- *Objetivo.* Describe lo que el actor desea lograr mediante la interacción con el sistema. Es el propósito o resultado esperado del caso de uso.
- *Flujo básico o principal.* Describe la secuencia principal de pasos que sigue el actor para lograr su objetivo con éxito. Este es el "camino feliz" o el flujo en el que todo sale según lo planeado.
- *Flujo o camino alternativo.* Son variaciones en el flujo principal que pueden ocurrir debido a diferentes condiciones o decisiones que toman los actores durante la interacción.
- *Precondiciones.* Son las condiciones que deben cumplirse antes de que el caso de uso pueda ejecutarse correctamente.

1. Nombre del Caso de Uso: Reservar Números de OC
2. Actores:

- **Responsable de Compras**: Persona autorizada para realizar la reserva de números de OC.

3. Descripción:
El **responsable de compras**, autorizado por el sistema, debe poder reservar un número de OC (Orden de Compra) para asegurar las operaciones de clientes especiales de la empresa. El sistema valida que existe la necesidad de la reserva y determina qué números están disponibles. Para ello debe quitar los números previamente reservados. Estos números quedan registrados en el sistema como reservados para su posterior utilización.
4. Precondiciones:

- El **responsable de compras** debe estar **autorizado** en el sistema.
- Debe existir una **necesidad de reserva de números de OC**.
- El sistema debe tener números de OC disponibles para reservar dentro del mes.

5. Flujo Principal (Escenario Básico):

- El **responsable de compras** ingresa al **Sistema Gestión de ATM**
- El sistema verifica que el usuario esté **autorizado** para realizar reservas.
- El sistema libera los números de OC reservados que no hayan sido utilizados en 15 días a partir de su reserva.
- El sistema calcula un rango de números de OC donde el mínimo es el "último número de OC utilizado +1" y 10 números posteriores disponibles (Ej. ultimo número de OC = 14, el sistema muestra de 15 a 25 siempre que en ese rango no existan números previamente reservados).
- El responsable de compras selecciona los números de OC.
- El sistema muestra los números reservados para su confirmación.
- El responsable de compras confirma la selección de los números de OC a reservar.
- El sistema registra responsable, fecha, hora y números de OC reservados, y muestra un **mensaje de éxito** al usuario.

6. Flujo Alternativo:

- **Actor no autorizado**:
  - Si el responsable de compras no está autorizado, el sistema mostrará un mensaje de error indicando que **no tiene permisos** para realizar la reserva. El caso de uso finaliza.
- **Liberar números de OC reservados**:
  - Si hay números de OC reservados desde hace 15 días o más, el sistema libera esos números para ser reutilizados.
- **No hay números de OC disponibles**:
  - Si no hay números de OC disponibles, el sistema permite liberar números reservados.

7. Postcondiciones:

- Los números de OC han sido reservados correctamente.
- La reserva se registra en el sistema correctamente.

8. Requisitos especiales:

- El sistema debe estar **disponible 24/7** para realizar la reserva.
- La operación no puede superar los 90 segundos
- El sistema debe garantizar la **seguridad** y el uso de **firmas digitales** para validar la operación.

9. Criterios de éxito:

- Los números de OC se reservan correctamente.
- El responsable de compras recibe una confirmación de éxito.
- El sistema genera un registro de la operación.

10. Criterios de aceptación:

- El responsable de compras solo puede realizar la operación si está autorizado.
- Los números de OC asignados deben ser **únicos** y **disponibles**.
- Se registra un historial de la reserva con información completa (responsable, fecha, hora, números de OC).

Figura 22 – Caso de uso

- *Postcondiciones.* Son los resultados esperados o el estado del sistema tras completar el caso de uso.
- *Flujo o camino de excepción.* Detallan lo que sucede si hay fallos o situaciones inesperadas durante la ejecución del escenario, como errores del sistema, datos inválidos, o comportamiento incorrecto del usuario.

En el contexto de los casos de uso, un *escenario* se refiere a una secuencia específica de pasos o eventos que representan una interacción particular entre un usuario (o actor) y el sistema. Son descripciones detalladas de los posibles caminos o flujos de interacción que pueden ocurrir cuando un actor interactúa con un sistema para lograr un objetivo específico, considerando tanto el *flujo principal* como las variaciones posibles (*alternativos* y *excepciones*). Permiten considerar alternativas para el diseño ya que ayudan a entender cómo se comportará el sistema en diferentes situaciones específicas. Sirven como base para diseñar pruebas funcionales que validen tanto los flujos principales como los alternativos y de error. Facilita la comunicación entre los involucrados/as, ingenieros de requisitos y diseñadores/as asegurando que se contemplen diferentes situaciones.

8.4 Escenarios

Como ya se mencionó en el primer capítulo, los *escenarios* son narrativas estructuradas de situaciones del contexto. Si bien cada escenario describe una situación particular, ninguno de ellos es completamente independiente del resto de los escenarios, sino que por el contrario cada uno guarda una relación con los otros. Algunas particularidades de un escenario son las siguientes:

- Deben cumplir un objetivo específico.

- Ocurren en un contexto determinado.
- Tienen un principio y un fin donde los actores realizan sus acciones con los recursos especificados.
- Transcurren en un tiempo dado, sin interrupciones.
- Tienen restricciones que limitan el comportamiento.
- Puede contener caminos alternativos de acción.

Autores como Firesmith, Sutcliffe, Schneider y Weidenhaupt, tienden a seguir un enfoque *top-down* para construir escenarios, comenzando con una visión general del sistema y descomponiéndola en componentes específicos. Michael Jackson utiliza un enfoque más *bottom-up*, donde los detalles técnicos y los problemas específicos construyen la visión global del sistema.

Existen varios tipos de escenarios, tales como narrativas, storyboards, video mock-ups, prototipos escritos y otros. Entre las narrativas que utilizan el lenguaje natural está la de Leite et al. que se conforma de un conjunto de componentes que permiten describir cada situación particular del contexto. Estos componentes son el título, objetivo, contexto, recursos, actores, episodios y excepciones.

El *título*, el *objetivo*, el *contexto*, los *recursos*, los *actores* y las *excepciones* son declarativos, mientras que los *episodios* son un conjunto de sentencias que muestran descripciones operacionales de comportamiento de los actores en el escenario. El co*ntexto*, los *recursos* y los *episodios* pueden contener *restricciones*, las cuales pueden estar asociadas a requisitos no funcionales.

Cada componente se debe completar teniendo en cuenta:

- *Título.* Es el identificador del escenario. Debe ser una breve descripción que sintetiza el contenido del escenario.

- *Objetivo***.** Representa para qué se realiza el escenario. El objetivo se satisface en el curso de acción de los episodios.
- *Contexto***.** Determina dónde se realiza el escenario**.** Se descompone en a) ubicación geográfica, b) ubicación temporal y c) precondiciones. Cada componente puede ser expresado por unas o más sentencias simples vinculadas por los conectores "y" y "o". La ubicación geográfica es donde se realiza el escenario. La ubicación temporal indica alguna limitación de tiempo. Y las precondiciones determinan qué se requiere previo para poder realizar el escenario.
- *Actores*. Son las personas, organizaciones y el mismo sistema de software a construir que participan en los episodios.
- *Recursos*. Son objetos tangibles e intangibles que utilizan los actores para realizar las acciones.
- *Episodios***.** Son las acciones que realizan los actores. Se representan a través de sentencias simples, condicionales u opcionales. Es importante destacar que los episodios pueden ser escenarios, posibilitando relaciones de jerarquía entre escenarios y sub-escenarios. Esta relación de jerarquía surge naturalmente cuando existen situaciones más pequeñas dentro de una situación mayor.
- *Excepciones***.** Son casos alternativos que suceden en algún momento del escenario y no permiten alcanzar el objetivo del escenario.

Título: Reservar Números de OC
Objetivo: reservar números de OC
Contexto:
Ubicación Geográfica: Oficina de compras
Ubicación Temporal:
Precondiciones: Debe existir una necesidad de reservar números de OC.
Actores: responsable de compras (Restricción: debe estar autorizado), sistema
Recursos: números de OC
EPISODIOS

1. El responsable de compras accede al ítem "Reservar Números de OC".
2. El sistema libera los números de OC reservados cuando han pasado 15 días o más.
3. El sistema calcula un rango de números de OC donde el mínimo es el "último número de OC utilizado +1" y 10 números posteriores disponibles (Ej. ultimo número de OC = 14, el sistema muestra de 15 a 25 siempre que en ese rango no existan números previamente reservados).
   Restricción: La operación no puede superar los 90 segundos.
4. El sistema muestra en pantalla el rango disponible.
5. El responsable de compras selecciona los números de OC que desea reservar.
   Restricción: se debe respetar el máximo de reservas por mes
6. El sistema muestra los números de OC seleccionados y solicita que se confirme o rechace la reserva.
7. Si el responsable de compras confirma la reserva entonces el sistema registra el responsable, fecha, hora y números de OC reservados.
8. Si el responsable de compras confirma la reserva entonces el sistema muestra un mensaje que la operación finalizó con éxito

Excepciones:
Ex1 → Causa: El responsable de compras necesita más números de OC que los disponibles. Solución. El sistema le permite al responsable de compras liberar reservas previas.

Figura 23 - Escenario

Es importante resaltar el poder de contextualización de los escenarios. Las situaciones se describen incorporando todas las acciones involucradas (episodios) y la única condición es estar en el alcance del objetivo del escenario, pero no necesariamente en el alcance del *objetivo del sistema*. Esto permite darle al escenario una amplitud que facilita comprenderlo en el contexto real. A tal punto es importante está ampliación del comportamiento, que pueden permanecer y evolucionar hasta el momento de la especificación, donde son excluidos. Por ejemplo, en el contexto del Sistema Gestión de ATM (ver Anexo), el arribo del ATM a Argentina no se incluye en esta versión del sistema,

pero pueden incorporarse algunos escenarios que lo describan para comprender su relación con la llegada y registro del ATM al depósito.

8.5 Diferencia entre casos de uso y escenarios

Los casos de uso son una técnica ampliamente utilizada para capturar y describir los requisitos funcionales de un sistema. Normalmente, utilizan un lenguaje estructurado y técnico. Con ellos se puede establecer un puente entre el análisis de requisitos y el diseño del sistema.

Los escenarios son una herramienta para representar contextos de uso con un enfoque narrativo y exploratorio. Describe contextos complejos sin necesidad de estructurar detalles técnicos o flujos de interacción específicos. Exploran y detallan situaciones del mundo real, identifican problemas y conflictos mientras se validan ideas en fases tempranas del desarrollo.

Se puede observar que los casos de uso se utilizan en proyectos con un fuerte enfoque en la *funcionalidad técnica* centrándose en **qué hace el sistema** y **cómo lo hace**, mientras que los escenarios en proyectos donde el *contexto de uso* es clave centrándose en **por qué** y **cómo** se usa el sistema en el mundo real. Estas diferencias se pueden observar comparando la Figura 22 y la Figura 23 donde se desarrolla el mismo ejemplo utilizando ambos modelos.

8.6 Requisitos implícitos y explícitos

Los modelos previamente descriptos pueden contener *requisitos implícitos* y *explícitos.* Los requisitos implícitos no están expresados directamente, sino que se derivan de la información que contiene el modelo y una vez identificados se deben especifica. Por ejemplo, un

requisito implícito en un sistema de gobernanza podría ser la seguridad y privacidad de los datos. Dentro de estos requisitos existen *acciones implícitas del sistema* que son expresiones que identifican acciones propias del sistema que no están documentadas con la precisión esperada. En este caso se deben extraer del modelo y completarlas para una correcta especificación.

Acciones implícitas del sistema		**Requisitos explícitos**	
Escenario	**Componente**	**ID**	**Descripción**
Reservar Números de OC	Episodio 1	REQ-001 **(RF)**	El Sistema debe permitir reservar números de OC.
	Episodio 2	REQ-002 **(RF)**	El sistema debe liberar los números de OC reservados cuando han pasado 15 días sin uso.
	Episodio 3 y Episodio 4	REQ-003 **(RF)**	El sistema debe calcular un rango de números de OC donde el mínimo es el "último número de OC utilizado +1" y 10 números posteriores disponibles (Ej. ultimo número de OC = 14, el sistema muestra de 15 a 25 siempre que en ese rango no existan números previamente reservados) y mostrarlo el resultado en pantalla.
	Restricción del Episodio 3	REQ-084 **(RNF)**	El sistema debe asegurar que la operación de reserva no supere los 90 segundos.
	Episodio 5	REQ-004 **(RF)**	El sistema debe permitir que el responsable de compras seleccione los números de OC que desea reservar.
	Restricción del Episodio 5	REQ-005 **(RF)**	El sistema debe controlar que no se superen por mes las reservas permitidas de números de OC.
	Episodio 6	REQ-006 **(RF)**	El sistema debe mostrar en pantalla los números de OC seleccionados durante una reserva.
	Episodio 6	REQ-007 **(RF)**	El sistema debe solicitar confirmación cada vez que se realice una reserva de OC.
	Episodio 7	REQ-008 **(RF)**	El sistema debe registrar el responsable, fecha, hora y números de OC reservados.
	Episodio 8	REQ-009 **(RF)**	El sistema debe mostrar un mensaje cuando la operación de reserva finalizó con éxito.
	Excepción1	REQ-010 **(RF)**	El sistema debe permitir al responsable de compras liberar números de OC reservados previamente.

Tabla 6 – Tipos de requisitos en un escenario

Los requisitos explícitos o específicos son aquellos que ya están

descriptos con todo el detalle necesario en el modelo y solo resta enviarlos a la ERS. Cuando el modelo incluye solo requisitos explícitos, el mismo modelo puede ser utilizado como documento de especificación.

En la Tabla 6 se muestra un ejemplo de cómo se transforman las *acciones implícitas del sistema* de un escenario a requisitos explícitos. Este ejemplo corresponde al escenario descripto en la Figura 23. Se puede observar que la restricción del episodio 5 se transforma en un RF; el episodio 6 se transforma en dos RF y la excepción del escenario en otro RF. Finalmente, los requisitos descriptos en la columna "Requisitos explícitos" formarán parte del documento de especificación donde serán analizados en conjunto, por ejemplo, para detectar duplicaciones.

Referencias

[Berry04] Berry D. M. and Kamsties E. (2004). "Ambiguity in requirements specification". En el libro de Prado Leite, J.C.S., Doorn, J.H. (eds) Perspectives on Software Requirements. Kluwer Academic Publishers, EEUU, ISBN: 1-4020-7625-8, Capítulo 2.

[Berr08] Berry, D.M. (2008). "Ambiguity in Natural Language Requirements Documents". In: Paech, B., Martell, C. (eds) Innovations for Requirement Analysis. Notes in Computer Science, vol 5320. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-89778-1_1

[Bexy06] Alfonso, Bexy. (2006). "Consideraciones relacionadas con las inspecciones de software". Ingeniería Industrial.

[Booc99] Booch, G., Rumbaugh, J., Jacobson, I. (1999). "The Unified Modeling Language User Guide", Addison-Wesley, Reading, MA.

[Cock00] Alistair Cockburn, (2000). "Writing Effective Use Cases", (1st. ed.). Addison-Wesley Longman Publishing Co., Inc., USA.

[Cohn04] Cohn, M. (2004). “User stories applied: For agile software development”, Addison-Wesley.

[Dano97] Dano, B., Briand, H., Barbier, F. (1997). "An Approach Based on the Concept of Use Case to Produce Dynamic Object-Oriented Specification", RE95: Proceedings of the Third IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, Los Alamitos, California, pp 54-64.

[Daou12] Norman Daoust. (2012). “UML Requirements Modeling For Business Analysts”, First Edition, Technics Publications, LLC, ISBN 13 -978-1935504245.

[Faga76] Fagan, M.E. (1976). “Design and Code Inspections to reduce Errors in Program Development”, IBM Systems Journal, Vol.15, Nº3, pp.182-211.

[Gane77] Gane, C., Sarson, T. (1977). “Structured Systems Analysis: Tools and Techniques”, Prentice-Hall, Englewood Cliffs, NJ.

[Gerv98] A. Gervasi, y B. Nuseibeh (1998). “Managing Inconsistent Specifications: Reasoning, Analysis and Action”, ACM Transactions on Software Engineering and Methodology, Vol. 7, Nº 4, pp.335–367.

[Gerv05] V. Gervasi (2005). “Reasoning about Inconsistencies in Natural Language Requirements”, ACM Transactions on Software Engineering and Methodology, Vol.14, Nº3, pp. 277-330.

[Jaco92] Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G.,Reading (1992). “Object-oriented Software Engineering - A Use Case Driven Approach”, MA: Addison Wesley, Nueva York: ACM Press.

[Jaco99] Jacobson, I., Booch, G., Rumbaugh, J. (1999). “The Unified Software Development Process”, Addison-Wesley, Reading, MA, 1º edición.

[Leit05] Leite J.C.S.P., Doorn J.H., Hadad G.D.S. y Kaplan G.N. (2005). “Scenario Inspections”, Requirements Engineering Journal, Vol.10, Nº1, Springer-Verlag, pp. 1-21.

[Patt14] Patton, Jeff (2014). “User Story Mapping”, O’Reilly, ISBN: 9781491904909.

[Ryan15] Ryan Mike, Wheatcraft, Louis, Dick, Jeremy and Zinni, Rick. (2015). “On the Definition of Terms in a Requirements Expression”, INCOSE International Symposium. 25. 10.1002/j.2334-

5837.2015.00055.x.

[Schn98] Schneider, G., Winters, J. (1998). “Applying Use Cases, A Practical Guide”, Addison-Wesley, Reading, MA.

[Weid98] Weidenhaupt, K., Pohl, K., Jarke, M., Haumer, P. (1998). “Scenarios in System Development: Current Practice”, IEEE Software, pp.34- 45.

Capítulo

9 Un enfoque centrado en procesos

9.1 Estrategia de la Ingeniería de Requisitos

La Ingeniería de Requisitos presenta una estrategia estructurada para la identificación, análisis, documentación, validación y gestión de los requisitos de un sistema de software, siendo crucial para asegurar su calidad. Una buena estrategia ayuda a reducir riesgos y mejorar el resultado final. Algunas ventajas de contar con una estrategia de requisitos son:

- *Mejora la comunicación y la colaboración*. Facilita la comprensión y el acuerdo mutuo sobre lo que debe lograrse.
- *Facilita la gestión de cambios.* Al tener requisitos documentados y aprobados, cualquier cambio necesario se puede gestionar de manera formal. Esto asegura que los cambios se evalúan en términos de impacto en el cronograma, costo y recursos, manteniendo el control sobre el proyecto.
- *Asegura la trazabilidad.* Vincula los requisitos con los objetivos, diseños, pruebas y entregables del proyecto, lo que permite un seguimiento efectivo del progreso y la validación de los resultados.

- *Alineación con los objetivos del negocio.* Una estrategia de requisitos asegura que los requisitos sean consistentes con los objetivos estratégicos de la organización. Permite priorizar los requisitos más importantes, alineando el desarrollo con el valor que aportan al negocio.
- *Facilita la toma de decisiones.* Proporciona información clara y estructurada que ayuda a las partes interesadas a tomar decisiones informadas durante todo el ciclo de vida del proyecto.
- *Reducción de riesgos.* Ayuda a mitigar riesgos relacionados con el incumplimiento de expectativas o la entrega de un producto que no satisface las necesidades.

La elección de una estrategia dependerá del tipo de proyecto, la naturaleza del sistema, las restricciones del entorno y la disponibilidad de recursos. Existen diferentes estrategias como la basada en el descubrimiento de requisitos, basada en modelos, de reutilización de requisitos, centrada en el usuario, basada en contextos específicos, dirigida por pruebas, etc.

A continuación, se describe una estrategia hibrida que se divide en tres etapas:

- Comprender el UdeD actual.
- Planificar el UdeD futuro.
- Documentar los requisitos del software.

Cabe mencionar que los verbos de cada etapa son autorreferenciales. Coincidiendo con la opinión de Parviainen et al. [Parv05] acerca de que no se puede modificar aquello que se desconoce, la estrategia comienza con *comprender* el contexto en estudio para luego *planificar* el proceso

del negocio con el sistema de software incluido. Como se puede observar en la Figura 24, en la primera etapa, *Comprender el UdeD actual,* se elicita y modela el proceso del negocio tal como existe al momento de iniciar la Ingeniería de Requisitos. Con este conocimiento se pueden tomar decisiones acerca de los servicios que deberá tener el software para satisfacer las necesidades del contexto. A partir del conocimiento generado en la primera etapa, se puede *Panificar el UdeD futuro*, que consiste en definir junto al cliente los servicios y restricciones que el software proveerá. La complejidad de esta etapa se debe a la necesidad de proyectar el futuro y pensar en un proceso del negocio que aún no existe.

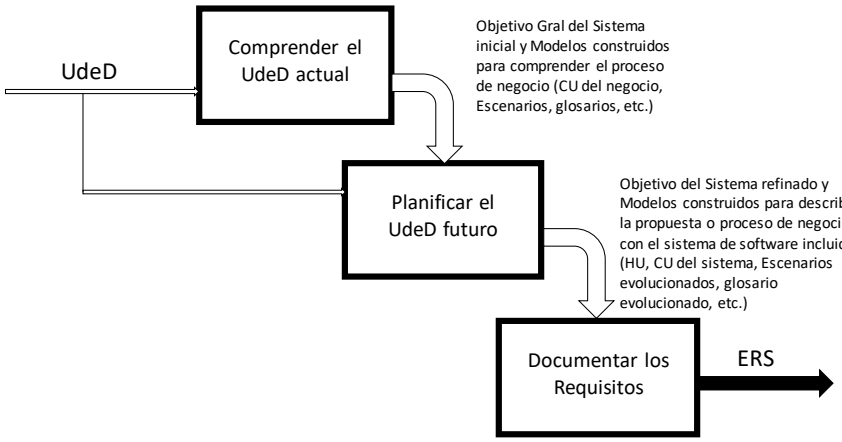


Figura 24- Estrategia de la Ingeniería de Requisitos

La construcción de modelos previos a la especificación permite identificar conflictos, inconsistencias, omisiones que pueden ser analizados con el cliente antes de obtener la especificación. Ayuda a definir claramente lo que está dentro y fuera del alcance del proyecto. Estos modelos previos contienen los requisitos inherentes y en la etapa *Documentar los requisitos* se extraen y se hacen explícitos en el documento de especificación de requisitos (ERS). El formato de este documento dependerá de las políticas organizacionales existentes, de

los estándares nacionales o internacionales que se utilicen, entre otras cuestiones.

9.2 Proceso de Requisitos

Para comprender qué es un proceso de requisitos se debe comprender primero qué es un proceso:

> **ISO 9001**, *"Un proceso es un "conjunto de actividades relacionadas o que interactúan, las cuales transforman elementos de entrada en resultados".*

El proceso de requisitos, como todo proceso, se compone de actividades (ver Figura 25) que generan resultados intermedios para obtener finalmente, el resultado esperado. Por ejemplo, un resultado intermedio podría ser la generación de casos de uso y el resultado final el producto software parcial o completo. En todo proceso se debe buscar la mayor eficacia y eficiencia posible para optimizar el tiempo y el costo.

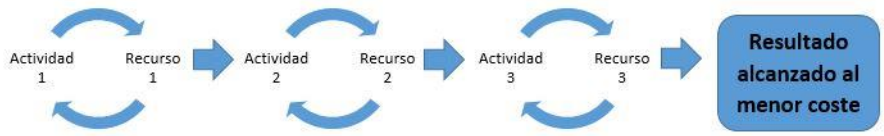


Figura 25 - El proceso descompuesto en actividades

Se puede observar en la Figura 26 que el *proceso de requisitos* tiene como entrada el conocimiento del dominio que contiene el proceso del negocio, en ocasiones ya existe el *objetivo del sistema* y alguna información para el nuevo sistema de software. Como salida se obtiene toda la documentación de los requisitos del software, particularmente la *Especificación de Requisitos de Software* (ERS) que representa los servicios y restricciones del nuevo sistema.

Figura 26 - El proceso de requisitos

Como se puede observar en la Figura 26, en algunos proyectos, la ERS puede estar acompañada de otros modelos. Existe una técnica que se denomina Visualización de Requisitos [Duan06] [Gote07] para permite crear representaciones visuales para transformar los requisitos en Diagramas de flujo, Mapas mentales, casos de uso, Wireframes, Modelos UML, Tableros Kanban o gráficos Gantt. Estas representaciones ayudan a comprender mejor los requisitos al traducir descripciones complejas en imágenes simples y comprensibles, facilitando el diálogo entre las partes interesadas, especialmente si no tienen experiencia técnica. Actúan como un recurso adicional durante todo el desarrollo del proyecto. Contar con un proceso de requisitos permite reducir malentendidos entre el equipo técnico y los involucrados/as, mejorar la colaboración y toma de decisiones, acelerar la definición de requisitos y mejorar la planificación del proyecto.

Se puede observar la diversidad de destinos de la ERS y los otros modelos, como ser clientes y usuarios, ingenieros de requisitos, equipo de diseño, equipo de pruebas y aseguramiento de calidad, líderes de proyecto, etc., lo que ha generado un importante interrogante para la disciplina:

¿Quién es el destinatario de la ERS?

La primera respuesta es que el principal destinatario de un documento de especificación es el grupo involucrados/as. Esto ha sido decisivo

para adoptar el uso intensivo del lenguaje natural, las técnicas de validación cercanas al usuario, las practicas colaborativas, etc. que le permite al cliente-usuario participar en la construcción de los requisitos. Pero también, la ERS es la columna vertebral de todo el proceso de construcción del software, por lo que es necesario que contenga aquella información, punto de vista o representación que asegure la comprensión de los requisitos por parte del grupo desarrolladores. Se puede comprender que la multiplicidad de destinos requiere de alternativas, pero deben asegurar en todo momento una absoluta consistencia entre la ERS y esos modelos.

9.3 Objetivos y sub-objetivos

Uno de los primeros elementos que se deben definir en un proceso de requisitos es el *objetivo del sistema.* Como ya se mencionó, en algunas ocasiones ya está definido en el UdeD y en otros, es necesario trabajar junto al cliente para precisarlo. El *objetivo del sistema* puede ser muy genérico al iniciar la Ingeniería de Requisitos ir refinándose a medida que se avanza. Un sistema de software puede tener más de un objetivo general, ya que los sistemas complejos suelen necesitar cubrir diversas necesidades y áreas funcionales. Estos objetivos se definen en función de las necesidades del negocio, los requisitos de los usuarios y los beneficios que debe proporcionar el software. Los objetivos representan las metas generales y de alto nivel que el software debe alcanzar. Los sub-objetivos son metas más específicas que descomponen un objetivo general en tareas o resultados más pequeños y manejables. Representan pasos o logros intermedios necesarios para alcanzar dicho objetivo. Cada sub-objetivo suele estar relacionado con una característica o funcionalidad específica del software. Por ejemplo, el *objetivo del sistema "Mejorar la eficiencia del proceso de facturación"* podría tener

como sub-objetivos *"Automatizar el cálculo de impuestos para reducir errores manuales" y "Generar automáticamente facturas electrónicas en formato PDF ".*

Para definir los objetivos y sub-objetivos se puede utilizar la técnica **SMART**:

- **S**pecific (específicos). Los objetivos deben ser claros y concretos.
- **M**easurable (medibles). Deben poder medirse para verificar si se han cumplido.
- **A**chievable (alcanzables). Deben ser realistas y logrables con los recursos disponibles.
- Relevant (relevantes). Deben ser importantes para los usuarios y la organización.
- **T**ime-bound (acotados en el tiempo). Deben tener un marco temporal definido.

Los objetivos y sub-objetivos deben revisarse de manera continua a lo largo del ciclo de desarrollo del software para ajustarlos si fuera necesario. Esto es crucial para tener una dirección clara y que el producto software satisfaga las expectativas del cliente. Estos objetivos ayudan a tomar decisiones durante el proceso de desarrollo, asegurando que todas las partes tengan una comprensión común de lo que se espera del nuevo sistema de software.

9.4 Nivel de cambio esperado

Detectar el *nivel de cambio* que va a sufrir el proceso del negocio (ver Figura 27) permitirá seleccionar el proceso de requisitos adecuado para cada proyecto en particular. En un contexto donde los cambios en el proceso del negocio son mínimos, la Ingeniería de Requisitos se torna

desmedida en términos de esfuerzo y costo. En este caso se sugiere un proceso de requisitos colaborativo para mantener una estrecha relación con los involucrados/as y capturar los requisitos en tiempo real, con reuniones frecuentes para revisar y ajustar los requisitos según las necesidades, con una mínima documentación.

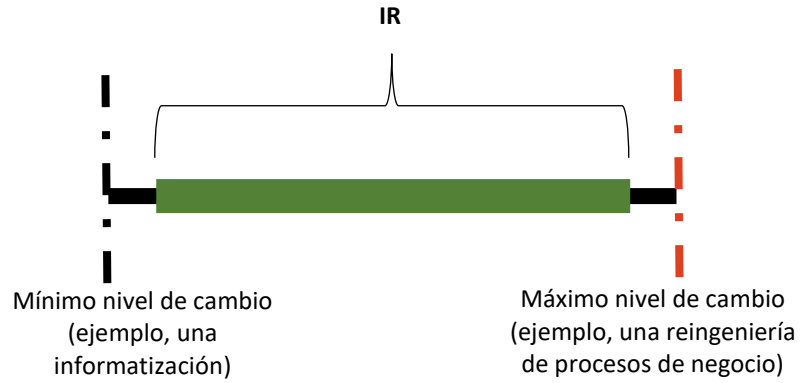


Figura 27 - Nivel de cambio del proceso del negocio

En el otro extremo, se encuentran los procesos de negocio con cambios radicales, donde se prevé una reingeniería de procesos del negocio[13]. En estos casos se debe seleccionar una estrategia de requisitos adecuada para acompañar la reestructuración organizacional esperada. Se sugiere un enfoque orientado a objetivos (Goal-Oriented Requirements Engineering) [Yu93] [Lapo05] o un enfoque ágil. La *Ingeniería de Requisitos*, normalmente estructurada y secuencial con un análisis riguroso de los requisitos y una documentación sólida, no suele ser la mejor opción en ninguno de los extremos mencionados, sino que es recomendable en sistemas medianos o grandes, con cierto grado de complejidad y con una cantidad de cambios en los requisitos controlables. Cabe mencionar que la elección de una estrategia puede ser mucho más compleja, ya que en ocasiones es necesario analizar un conjunto de variables como la naturaleza del proyecto, riesgos, tiempos

[13] Según lo propuesto por Michael Hammer y James Champy.

y plazos de entrega, entre otras, para determinar la estrategia óptima. Implementar una estrategia sólida de Ingeniería de Requisitos es esencial para lograr los resultados esperados.

9.5 Actividades de la Ingeniería de Requisitos

La Ingeniería de Requisitos [Pohl10] [Hull17] [Pres20] se conforma por un conjunto de actividades relacionadas entre sí que, a su vez, se pueden descomponer en sub-actividades. En la Tabla 7 se describen algunos procesos de requisitos y las actividades más importantes que los componen. Puede observarse que existen varias coincidencias, a pesar de que cada autor contempla una perspectiva propia de cómo se debe llevar a cabo el proceso.

Durante las primeras ejecuciones de estas actividades existe una secuencialidad en su ejecución, pero a medida que se avanza en el proceso de requisitos, se vuelven concurrentes. Por ejemplo, mientras se está validando es posible detectar algún defecto o elicitar nueva información.

Loucopoulos & Karakostas [Louc95]	**Thayer & Dorfman** [Thay97]	**Kotonya & Sommerville** [Koto98]	**Leite** [Leit94][Leit01]	**Sawyer & Kotonya** [Sawy01]
Elicitar Especificar Validar Gestionar	Elicitar Analizar Especificar Verificar Gestionar	Elicitar Analizar y Negociar Documentar Validar Gestionar	Elicitar Modelar Analizar Gestionar	Elicitar Analizar Especificar Validar

Tabla 7 - Actividades de la Ingeniería de Requisitos

Se puede observar en Tabla 7 que existen algunas diferencias en la definición de las actividades pero además, existen homónimos entre las

actividades como es el caso de *analizar* que para Leite corresponde a determinar la calidad de los requisitos mediante la verificación y la validación de los modelos y documentos generados; para Kotonya involucra construir modelos y verificarlos y para Thayer es modelar los requisitos. Otra diferencia es la *negociación* que está claramente diferenciada en Kotonya, mientras que para otros autores no se mencionada como una actividad separada.

A continuación, se describen las actividades propuestas por Leite:

- *Elicitar*[14]. Se refiere al proceso de identificar, descubrir y recopilar información. Inicia identificando las fuentes de información (FI) existentes en el dominio las cuales luego, se priorizan teniendo en cuenta, entre otras características, el tipo de información que proveen y la disponibilidad para acceder a ellas. Cada FI provee un tipo de información. Por ejemplo, los manuales de procedimientos tienen un punto de vista del "deber ser" y su accesibilidad de ilimitada. Las personas proveen diferentes tipos de información dependiendo de su ubicación en la pirámide organizacional. Cuando proviene del nivel más alto de la pirámide, se refiere por lo general, a objetivos y políticas organizacionales y su punto de vista es del "deber ser"; en el centro de la pirámide pueden convivir distintos niveles de detalle y aparecer tanto el "deber ser" como el "es"; finalmente, en la base de la pirámide la información es puntual sobre tareas específicas y predomina el punto de vista "es" (para ampliar ver Capítulo 10). Con las personas la disponibilidad debe ser definida al inicio de la IR para asegurar el proceso. Para obtener

[14] Nuseibeh and Easterbrook [Nuse00] hacen referencia que uno de los objetivos de la elicitación es descubrir qué problema debe resolverse.

información se utilizan diferentes técnicas de elicitación, las cuales tienen su origen en las Ciencias Sociales [Gogu93]. Durante la elicitación se debe asegurar que todo el conocimiento implícito existente se transforme en explícito, por ejemplo, al analizar diferentes perspectivas de un mismo tema.

- *Modelar.* Consiste en representar de forma gráfica, formal o estructurada los hechos recolectados durante la elicitación. Es fundamental para comprender, comunicar, reducir errores, tomar decisiones, facilitar el mantenimiento y la evolución del software, especialmente en proyectos complejos. Existen diferentes modelos que permiten resguardar la información del contexto, como UML (Unified Modeling Language), Diagramas de flujo, ERD (Entity-Relationship Diagram), Modelos conceptuales, Prototipos, entre otros. Para ampliar ver Capítulo 8.

- *Analizar.* Se trata de descubrir y resolver las inconsistencias, omisiones, ambigüedades, contradicciones y errores presentes en los modelos. Comprende las sub actividades verificar, validar y negociar. Las dos primeras son procesos esenciales para garantizar la calidad y confiabilidad de la información (para ampliar ver Capítulo 12). La negociación apare ante conflictos. Se debe tener en cuenta que es durante la definición de los RNF donde se encuentra la mayor cantidad de conflictos y es aquí donde la negociación incrementa su influencia. A modo de ejemplo, se pueden citar los siguientes conflictos de a pares: mantenibilidad-eficiencia o seguridad–disponibilidad [Koto98].

Es habitual que esta actividad encuentre un momento propio en el proceso de requisitos donde se despliega a pleno, pero puede ser concurrente o espontánea con otros momentos, como la elicitación y la validación.

- *Gestionar.* Esta actividad comienza cuando se genera la primera versión de la ERS. Los requisitos son priorizados, rastreados, validados y gestionados durante todo el ciclo de vida del proyecto. Es fundamental para asegurar que el producto final cumpla con las necesidades y expectativas de los involucrados/as. En esta actividad se identifican, analizan y realizan los cambios en los requisitos. Estos cambios pueden ser generados por requisitos nuevos, modificaciones a requisitos existentes (errores, mejoras, etc.) o requisitos que evolucionan. Durante la Gestión de Requisitos es importante poder rastrear los requisitos del software en ambas direcciones, a través de todos los períodos de continuo refinamiento e iteración en cualquiera de sus etapas. Para ampliar ver Capítulo 13.

9.6 Elicitación vs. Modelado

Cada modelo utilizado en un proceso de requisitos es generado para satisfacer algún objetivo específico, por ejemplo, comprender el sistema desde una perspectiva funcional representando como se procesan los datos (DFD); descomponer un sistema complejo en componentes manejables permitiendo una comprensión detallada del sistema en términos de acciones y flujos de control (SADT); definir los requisitos centrándose en la interacción actor-sistema (casos de uso); definir los requisitos a través de las descripciones de situaciones del dominio (escenarios); etc. Naturalmente, el ingeniero de requisitos

intenta concentrarse en ese objetivo específico y completar el modelo con la mayor fidelidad posible. Pese a los muchos esfuerzos realizados, existe una limitación estructural en todos los procesos utilizados y que reside en la coordinación de las actividades de elicitación y modelado. La información capturada y los objetivos a alcanzar por el ingeniero de requisitos pueden no coincidir en su ordenamiento temporal. Es así que es necesario privilegiar uno de los órdenes temporales en desmedro del otro. En otras palabras, la coordinación de las actividades de elicitación y modelado puede realizarse de las siguientes maneras:

- *Modelado guiado por la elicitación*. El proceso se inicia en el UdeD elicitando información no estructurada, luego el ingeniero de requisitos debe decidir en qué modelo incorporar la información que obtuvo. No existe un control rígido sobre la elicitación. Esta forma de trabajar requiere de un gran manejo de todos los modelos disponibles y de una gran capacidad del ingeniero de requisitos para manipular información dispersa y modelos incompletos.
- *Elicitación guiada por los modelos.* La actividad de elicitación es controlada por el modelo a construir en cada momento del proceso de requisitos. El ingeniero de requisitos busca información para completar particularmente ese modelo.

Ambas formas de trabajar idealizadas tienen ventajas e inconvenientes. El enfoque *modelado guiado por la elicitación* asegura que toda la información adquirida sea representada en algún modelo, pero es altamente dependiente de la habilidad y experiencia del ingeniero de requisitos para asegurar el cubrimiento completo del problema bajo estudio. En el enfoque *elicitación guiada por los modelos* se tiene más

control sobre la elicitación. O sea, el ingeniero de requisitos va en busca de la información necesaria para completar un modelo en particular. La mayoría de los autores se enrolan implícita o explícitamente en un enfoque de elicitación guiada por los modelos.

9.7 Glosarios en los procesos de requisitos

Uno de los factores que afectan la calidad del software es la dificultad de establecer una comunicación segura entre todas aquellas personas que tienen alguna participación en la construcción de software. Una comunicación fallida puede provocar distorsiones, malos entendidos, contradicciones y omisiones no deseadas. El uso de glosarios en los procesos de requisitos tiene como propósito principal reducir la ambigüedad y mejorar la precisión, tanto en la comunicación escrita como oral. Diferentes autores han estudiado el impacto que tiene el vocabulario utilizado durante la construcción del software. Zave et al. [Zave97] afirma que "toda la terminología usada en Ingeniería de Requisitos deberá estar fundada en la realidad del ambiente para el cual se construirá una máquina", además, sostiene que cada término debe estar descripto para asegurar su uso correcto. Weidenhaupt et al. [Weid98] muestra un estudio donde se analizaron quince proyectos industriales, de los cuales cuatro usaron glosarios. El objetivo de estos glosarios fue procurar una mejor comunicación. Ben Achour et al. [BenA99] determinó que la mitad de los casos de uso de un proyecto tenían errores basados en la falta de comprensión de la terminología utilizada, acentuando la necesidad de contar con glosarios que desambigüen el léxico utilizado.

Es importante remarcar que en la construcción de software en general, está muy recomendado el uso de glosarios. Pueden ser construidos tempranamente para enriquecer todo el proceso y garantizar una comunicación exitosa. En otros casos, son construidos junto a la ERS para desambiguar particularmente este documento.

Referencias

[BenA99] Ben Achour, C., Rolland, C., Maiden, N.A.M., Souveyet, C. (1999). "Guiding Use Case Authoring: Results of an Empirical Study", International Symposium On Requirements Engineering (RE´99), Limerick, Irlanda, IEEE Computer Society Press, pp.36-43.

[Duan06] Duan, C. and Cleland-Huang, J. (2006). "Visualization and Analysis in Automated Trace Retrieval", First International Workshop on Requirements Engineering Visualization, REV'06.

[Gogu93] Goguen, J.A., Linde, Ch. (1993). "Techniques for Requirements Elicitation", IEEE First International Symposium on Requirements Engineering, RE'93, IEEE Computer Society Press, Los Alamitos, CA, pp.152-164.

[Gote07] Gotel O.C.Z., Marchese F. T. And Morris S. J. (2007). "On Requirements Visualization", Second International Workshop on Requirements Engineering Visualization, REV 2007.

[Hull17] Hull, E., Jackson, K., & Dick, J. (2017). "Requirements engineering" (4th ed.). Springer.

[ISO9001] ISO 9001:2015 (2015). "Sistema de Gestión de Calidad - Requisitos".

[Koto98] Kotonya, G., Sommerville, I. (1998). "Requirements Engineering: Processes and Techniques", John Wiley & Sons.

[Lapo05] Lapouchnian, A. (2005). "Goal-Oriented Requirements Engineering: An Overview of the Current Research". https://sg1.run/xs

[Leit94] Leite, J.C.S.P. (1994). "Engenharia de Requisitos", Notas Tutoriales, material de enseñanza en el curso Requirements Engineering, Computer Science Department of PUC-Rio, Brasil.

[Leit01] Leite, J.C.S.P. (2001). "Gerenciando a Qualidade de Software com Base em Requisitos", en el libro Qualidade de Software: Teoria e Prática, editores A.R. Rocha, J.C. Maldonado y K.C. Weber, Prentice-Hall, San Pablo, capítulo 17, pp.238-246.

[Louc95] Loucopoulos, P., Karakostas, V. (1995). "System Requirements Engineering", McGraw-Hill, London.

[Nuse00] B. Nuseibeh and S. Easterbrook. (2000). "Requirements engineering: A roadmap". In A. C. W. Finkelstein, editor, The Future of Software Engineering, Limerick, Ireland. IEEE Computer Society Press. (Companion volume to the proceedings of the 22nd International Conference on Software Engineering, ICSE'00).

[Parv05] Parviainen, P., Tihinen, M., van Solingen, R. (2005). "Requirements Engineering: dealing with the Complexity of Sociotechnical Systems Development", in Requirements Engineering for Sociotechnical Systems, J.L. Maté & A. Silva (eds.), Chapter I, pp1-20, ISBN 1-59140-506-8.

[Pohl10] Pohl, K. (2010). "Requirements engineering: Fundamentals, principles, and techniques". Springer.

[Pres20] Pressman, R. S., & Maxim, B. R. (2020). "Software engineering: A practitioner's approach" (9th ed.). McGraw-Hill Education.

[Sawy01] Sawyer, P., Kotonya, G., (2001). "Software Requirements", SWEBOK, Guide to the Software Engineering Body of Knowledge, IEEE Computer Society, capítulo 2, pp.31-56, IEEE Trial Version 1.00, http://www.swebok.org/stoneman/ trial_1_00.html

[Thay97] Thayer R.H., Dorfman, M. (1997). "Software Requirements Engineering", IEEE Computer Society Press, 2º edición, Los Alamitos, CA.

[Weid98] Weidenhaupt, K., Pohl, K., Jarke, M., Haumer, P. (1998). "Scenarios in System Development: Current Practice", IEEE Software, pp.34- 45.

[Yu93] Yu, E. (1993). "Modeling organizations for information systems requirements engineering", IEEE First International Symposium on Requirements Engineering, IEEE Computer Society Press, San Diego, pp.34-41.

[Zave97] Zave, P., Jackson, M. (1997). “Four Dark Corners of Requirements Engineering”, ACM Transactions on Software Engineering and Methodology, Vol.6, Nº1, pp.1-30.

Capítulo

10 Información extemporánea

En el marco de la elicitación guiada por los modelos el ingeniero de requisitos debe seleccionar las técnicas de elicitación que le aseguren que la información obtenida es la esperada para completar el modelo que está construyendo. En este contexto, suele aparecer información que no puede ser representada en ese modelo particular. Aunque encontrará un modelo donde se pueda alojar, el ingeniero de requisitos se enfrenta a un dilema que debe resolver:

¿qué hacer con esa información?

En general, esta información no esperada suele ser registrada en diferentes tipos de anotadores y en documentos no estructurados. Existen dos factores que agravan las desventajas de los documentos no estructurados. Estos factores también inciden en los documentos semi estructurados, pero pueden ser fácilmente controlados al ser considerados de una manera ordenada. Estos factores son los siguientes:

- El tiempo.
- La contextualización de la información.

El primer ítem se refiere al tiempo transcurrido desde que la información no esperada aparece hasta que es incluida en el modelo que la debe contener. Este tiempo varía significativamente si el modelo donde debe ser incorporada existe o aun no fue creado. Puede suceder que en ambos casos transcurra el mismo tiempo, pero cuando el modelo existe se espera que su inclusión sea inminente al momento de su aparición. En cambio, cuando el modelo no existe este tiempo es relativo dependiendo de cuando se construya dicho modelo. Este tiempo puede ser un factor desestabilizador ya que el tiempo actúa como un neutralizador de la información descripta. Es así que se olvidan algunos elementos que parecían obvios en el momento de la aparición. Se supone que en un corto plazo estos elementos se encuentran todavía disponibles. En este momento ya se describe el segundo ítem que corresponde a la contextualización de la información no esperada. O sea, en el momento en el cual aparece se encuentra relacionada con otra información que no queda registrada. Esta omisión puede generar ambigüedad o directamente que no se comprenda. Esto depende de la complejidad de la información no esperada, de su repercusión en el resto de la información elicitada y del detalle utilizado para describirla. Por todos estos motivos, la información no esperada requiere ser registrada e incorporada al proceso de requisitos que se esté utilizando. En el momento oportuno, esta información puede ser descartada por ser irrelevante para el nuevo sistema de software o contener aspectos importantes. Un tratamiento inadecuado puede convertirse en un error en la definición de la solución con un ulterior aumento del costo del software. Esta información no esperada se la denomina *Información Extemporánea (iE)* [Kapl09] [Kapl14] [Kapl15]. Se puede definir una taxonomía para la iE relacionada con la

temporalidad de su aparición. Por un lado, cuando el modelo donde debe ser incorporada ya está construido, se la denomina *información tardía* (iT). El caso opuesto, cuando el modelo que la debe alojar aún no fue construido, se la denomina *información adelantada* (iA). En este último caso, se desconocen las consecuencias que puede tener sobre la nueva información a elicitar y si presenta riesgo de perderse, de ser olvidada o de ser incorrectamente comprendida. En la Figura 28 se representa cualitativamente la cantidad de iA y de iT durante la estrategia de requisitos presentada en el capítulo anterior. Estos valores son sólo representativos.

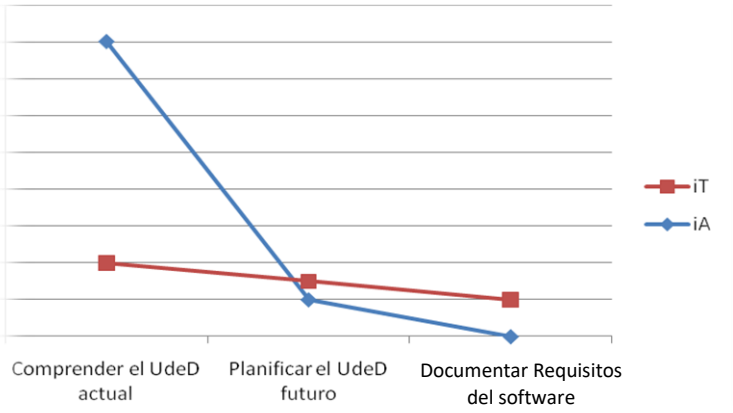


Figura 28- iE en cada etapa de la estrategia de requisitos

Cabe destacar que usualmente esta información es particularmente útil cuando se encuentra en manos del ingeniero de requisitos y es su responsabilidad incorporarla en los modelos pertinentes. Es también el ingeniero de requisitos quien debe determinar si es necesario registrar esa información para su posterior uso, teniendo en cuenta dos variables: la importancia de la información en el contexto en estudio y la posibilidad de que no vuelva a aparecer en todo el proceso de requisitos.

10.1 Información adelantada

La preponderancia de las entrevistas sobre otras técnicas de elicitación en relación con la iA es notoria y visible pero no excluyente. Por ejemplo, la lectura de bibliografía del dominio (manuales de procedimientos, formularios, etc.) es una fuente de información muy utilizada y rica para modelar, pero suele tener algunas menciones, tal vez desperdigadas, de aspectos necesarios para construir la solución. La condición de desperdigada es relevante en este contexto ya que, si el documento tiene información relacionada con la solución, pero concentrada, meramente habrá que registrar este hecho para retomar la lectura del documento cuando sea el momento adecuado. Pero cuando está desperdigada el riesgo de omisión crece en forma inversa con el tamaño de los fragmentos existentes. Otro ejemplo de aparición de iA se presenta al utilizar la técnica de observación para capturar conocimiento, el ingeniero de requisitos puede enfrentarse a un suceso claramente relacionado con un modelo a ser confeccionado más adelante, pero carece de la certeza de poder volverlo a observar cuando atienda ese modelo, haciendo necesario su registro adelantado. Asimismo, el ingeniero de requisitos puede tener ideas potencialmente muy apropiadas en relación con lo que escucha, observa y lee. Esto está relacionado con la memoria de corto plazo, la que le suele jugar malas pasadas a los creativos en cualquier actividad. Lamentablemente muy poco tiempo después de haberse concebido una idea sólo se recuerda precisamente eso, que se tuvo una idea, pero no se recuerda exactamente cuál. Esto cae completamente en el paradigma de la iA, aunque su origen no fue ni el cliente ni el usuario. Por otra parte, es muy difícil determinar en una etapa temprana, si la información elicitada o las ideas del ingeniero de requisitos son válidas; si contienen

la raíz de lo que luego serán requisitos o sencillamente son aspiraciones no aplicables en el contexto futuro. La correcta manipulación de la iE debe afrontar el problema de que el ingeniero de requisitos está tratando de concentrarse en su objetivo principal y particularmente la iA lo perturba en su trabajo, de manera que se corre el riesgo de no hacer ni una cosa ni la otra. Por lo tanto, se debe minimizar esa perturbación y al mismo tiempo aprovechar la iA tanto como sea posible. El problema así visualizado es muy similar al recorrido de un grafo con bifurcaciones, lo que se hace en este caso es seguir el camino principal asegurándose poder retornar al camino alternativo cuando sea oportuno. Es decir, el ingeniero de requisitos debe conservarse lo más apegado posible a la actividad principal para lograr el objetivo deseado, pero al mismo tiempo debe encontrar una forma de recordar lo mínimo necesario para retornar al camino secundario posteriormente. Si bien la iE molesta al ingeniero de requisitos no se debe intentar eliminarla ya que su existencia puede ser un factor para asegurar la calidad de los requisitos del software. La iA en particular evidencia la presencia de muchas ideas que pueden enriquecer el conjunto de requisitos de software. La presencia de iA puede ser un hecho cuasi-anecdótico o ser un problema significativo en un proyecto de desarrollo de software (ver

La información adelantada aparece cuando:	
el usuario…	***el ingeniero de requisitos…***
• describe algún problema al realizar una tarea; • menciona necesidades insatisfechas; • describe expectativas para el futuro; • sabe que necesita o requiere del nuevo sistema de software.	• recurre a su experiencia y conocimiento para encontrar solución a problemas, necesidades o expectativas de los usuarios; • recurre a su experiencia con otros sistemas de software similares para proponer funcionalidades alternativas.

Tabla 8). La cantidad de información a registrar puede verse multiplicada por la cantidad de ingenieros de requisitos y de fuentes de

información involucradas en el proceso. Puede verse agravada por la existencia de múltiples puntos de vista y por la complejidad del dominio. El registro semiestructurado pretende distraer lo menos posible la atención del ingeniero de requisitos asegurando el menor grado de distorsión de la línea principal de trabajo [Perl75], ofreciendo una forma simple de registrar esta información, con un marco homogéneo y ordenado y principalmente dejando la información en forma clara y visible para ser utilizada en el momento oportuno.

La información adelantada aparece cuando:	
el usuario…	***el ingeniero de requisitos…***
• describe algún problema al realizar una tarea; • menciona necesidades insatisfechas; • describe expectativas para el futuro; • sabe que necesita o requiere del nuevo sistema de software.	• recurre a su experiencia y conocimiento para encontrar solución a problemas, necesidades o expectativas de los usuarios; • recurre a su experiencia con otros sistemas de software similares para proponer funcionalidades alternativas.

Tabla 8 - Principales situaciones donde aparece iA

La iA está asociada a dos características que combinadas conforman su perfil en el contexto. Una es cuantitativa y está relacionada con la *expectativa del cliente-usuario* mientras que la otra es cualitativa y tiene que ver con el *rol del entrevistado dentro de la organización*.

10.2 Expectativa del cliente-usuario

La expectativa del cliente-usuario es determinante para la aparición de iA y es el factor más significativo referido a la cantidad de iA que puede aparecer. Cuando se entrevista a personas que esperan/necesitan alguna solución del nuevo sistema de software, como se puede observar en la Figura 29, se obtiene mayor cantidad de iA, mientras que, en el otro extremo, cuando el entrevistado no tiene ninguna expectativa con

respecto al nuevo software y, en algunos casos, ni siquiera sabe que puede esperar del mismo, la aparición de iA es prácticamente nula.

Se debe considerar también que la cantidad de iA aumenta considerablemente según la cantidad de cambios que se planifica realizar en el proceso del negocio. En la Tabla 9 se enumeran algunos factores que pueden desencadenar iA.

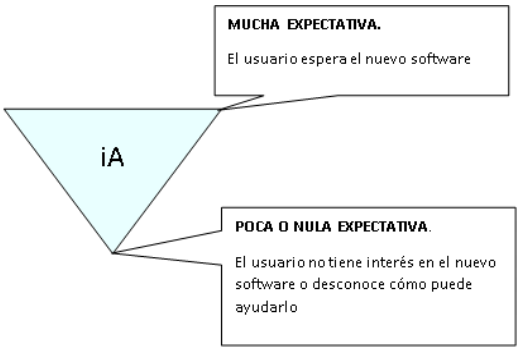


Figura 29 - Cantidad de iA según la expectativa del cliente-usuario

Factores Internos	Factores Externos
• Incorporación de Gestión de Calidad; • Tercerización de actividades; • Apertura de nuevas líneas de negocio; • Cambios en la tecnología de producción; • Aumento en los volúmenes de ventas; • Cambios en los objetivos organizacionales; • Cambios en los productos o servicios • Cambios socio-culturales que afectan a la organización.	• Cambios económicos y políticos que afectan el negocio; • Cambios en las preferencias de los clientes; • Legislación nueva.

Tabla 9 - Factores que influyen en la aparición de iA

10.3 Rol del entrevistado dentro de la organización

Las entrevistas para construir un sistema de software se deben planificar de tal manera que todos los puntos de vista de la organización estén incluidos. Se analiza el tipo de información de

acuerdo al rol del entrevistado en la pirámide organizacional. Cuando se entrevista a personal jerárquico (directores, gerentes, jefes) se obtiene iA referida en su mayoría a objetivos, metas y aspectos no funcionales, con un alto grado de abstracción. La mirada es estratégica y de toda la organización. El personal que tiene responsabilidad en la parte media de la pirámide tiene responsabilidades tácticas y transfiere conocimiento acerca de sub-objetivos o aspectos funcionales y no funcionales de coordinación y de toma de decisiones en sectores o áreas particulares. Cuando se entrevista a personal operativo, quien realiza una determinada tarea, la iA obtenida es más precisa y detallada. O sea, que el tipo de información obtenida en cada nivel de la pirámide organizacional está relacionado con sus actividades.

Figura 30 - Tipo de iA según su origen

Indudablemente la iA producida por cualquier fuente de información o eventualmente por el ingeniero de requisitos proviene de un proceso cognitivo y sería muy apropiado registrar los mecanismos que dieron lugar a ese conocimiento, pero esto es a todas luces una tarea casi imposible o por lo menos impráctica ya que en la casi totalidad de los casos el mecanismo de concepción de esta iA es desconocida.

10.4 Información tardía

La iT se presenta de manera similar a la iA. Sucede durante la elicitación de conocimiento cuando un cliente-usuario especifica o corrige información existente o cuando el ingeniero de requisitos descubre alguna inconsistencia mientras elicita o descubre una nueva interpretación de algún aspecto comprendido incorrectamente. Esta información es, en general, una aclaración, una explicación, una observación o una modificación al conocimiento que se está elicitando pero que afecta un conocimiento ya modelado. Que el modelo en el que debe ser incluida exista es lo que la caracteriza como información tardía. Este modelo ya construido disminuye sensiblemente el nivel de preocupación del ingeniero de requisitos respecto a la posibilidad de perderla u olvidarla, ya que su incorporación puede ser en un plazo corto de tiempo. El peligro más serio de la iT es que contenga conocimiento importante que no fue pensado ni modelado, haciendo repensar o remodelar parte de lo construido.

10.5 Información extemporánea en los procesos de requisitos

La iE, particularmente la iA, debe ser integrada a toda la información elicitada. De esta manera se garantiza una mayor completitud de los modelos a construir que describen la solución esperada por el cliente. La mera posibilidad de perder información valiosa es suficiente para alertar sobre su existencia y cuidarla durante todo el proceso de requisitos. Esta inclusión debe cumplir dos premisas básicas: que genere una mínima distracción para el ingeniero de requisitos y que pueda ser correctamente recuperada cuando sea el momento oportuno. La mínima distracción significa no desviar la atención del ingeniero de

requisitos del modelo que se está construyendo, ya que ese es su principal objetivo. Para ello se puede generar una ficha de información extemporánea (FiE) que permita registrar la iE en el mismo momento que aparece. Se debe recordar que la iE se utiliza para corregir errores en modelos construidos (iT) y como información relevante cuando se debe definir la solución para el nuevo sistema de software (iA). El tratamiento de la iE tiene tres momentos significativos:

- Resguardo de la iE.
- Análisis de la iE.
- Cierre de la iE.

El *resguardo de la iE* se puede realizar utilizando una ficha de apertura como se puede observar en la Figura 31.

FICHA DE INFORMACION EXTEMPORÁNEA

Datos de Apertura

Identificación FiE: 007

Proyecto: *Administración de ATM* **Fecha**: 10/05/2024

Origen: FI √ - Ingeniero de Req. **Nombre del Origen**: *Jefe de Compras*

Tipo iE : iA √ - iT **Nivel de impacto**: Alto - Medio √ - Bajo

Tratamiento urgente: Si – NO √ **Dispersión**: Si – NO √

Incluir en el Modelo: Escenarios Futuros **Ítem**: Reservar Números de PO

Descripción de iE: es necesario reservar algunos números de PO para clientes especiales (gobierno) con los que se tiene un acuerdo previo de fabricar cierta cantidad ATM en un tiempo determinado y se sabe que serán necesario generar esas PO.

Comentario: ……

……

Figura 31 - Ejemplo de apertura de la FiE

A cada FiE se le debe asignar un número univoco que puede ser un número secuencial, pero debe permitir identificar la ficha durante todo el proceso. Se deben completar los datos del proyecto, fecha de

apertura, origen, nombre del origen, tipo de iE, nivel de impacto, urgencia, dispersión y datos acerca de donde se sospecha que debe ser incluida (modelo e ítem). Todos estos datos tienen como objetivo contextualizar la iE. Dos aspectos se deben tener en cuenta para su tratamiento: la dispersión y la fragmentación. Reconocer si tiene dispersión es muy importante porque determina si esa iE aplica o puede aplicar a más de un modelo o ítem (ejemplo, un aspecto de seguridad se dispersará en distintos requisitos). También la iE puede requerir ser fragmentada en varias fichas cuando la iE es muy general. El componente descripción de la iE es el más relevante y, por lo tanto, para asegurar su comprensión se sugiere que sea una descripción completa, clara pero lo más concreta posible. De ser necesario, es posible agregar un comentario para dejar sentado algún dato adicional. Es importante mencionar que en el momento que aparece la iE existe una presunción bastante acertada sobre todos los datos que se deben incorporar a la ficha. Esto sucede por la cercanía del momento de su aparición donde su contextualización es óptima. En la Figura 31 se puede analizar un ejemplo de apertura de la ficha. Los datos marcados con una tilde corresponden a lo que ha elegido el ingeniero de requisitos en el momento de la apertura. Esta forma de completar la ficha tiene el propósito de distraerlo lo menos posible. Cuando llega el momento de incluir la iE en el modelo correspondiente, se debe *analizar la iE*. Para este fin, se pueden agrupar las fichas por tipo (por ejemplo, toda la iA o toda la iT), por urgencia, por impacto, por modelo. El agrupamiento de las FiE ayuda a tenerlas disponibles para ser consultadas oportunamente. En ese caso, se analiza cada FiE y se decide dónde y cómo incluirla. También es importante tener visible toda información con dispersión para analizar su impacto. Es

importante destacar que el modelo e ítem puede coincidir con la ficha o no, es una decisión que tomará el ingeniero de requisitos en el momento de analizarla.

Simultáneamente a la inclusión, se debe *cerrar la FiE.* Para ello se completa otra parte de la ficha, como se puede observar en la Figura 32. Debe llevar el mismo número identificador de la apertura, la fecha actual y el responsable de la decisión. Si la FiE fue aceptada se completa el modelo y el ítem en el cual se incluyó. En ambos casos, aceptada o rechazada, se puede ingresar un comentario adicional para dejar sentado algún concepto aclaratorio. Una iE puede ser rechazada por entrar en conflicto con otro requisito o con una regla del negocio o por ser considera innecesaria. En el ejemplo de la Figura 32 se puede observar el cierre de la FiE número 007 cuya responsable es la Ingeniera María P. La ficha fue aceptada e incorporada en el escenario Futuro Reservar números de OC.

FICHA DE INFORMACIÓN EXTEMPORÁNEA

Datos de Cierre

Identificación FiE: 007

Fecha: 11/10/2024 Responsable: *Ingeniera María P.*

Estado de la FiE: Aceptada √ - *Descartada*

Modelo en el cuál se incluyó: *Escenarios Futuros*

Ítem: *Reservar Números de PO*

Comentario: ……

……

Figura 32 - Ejemplo de un cierre para una FiE

La ficha propuesta es solo a modo de ejemplo, se puede construir una ficha según las necesidades de cada proyecto. Como dato final, estas fichas también pueden ser utilizadas para la Gestión de Requisitos, ya que cuando ingresa un cambio el mismo debe ser evaluado para ser

aceptado o rechazado. En el caso de ser aceptado surge muy natural registrarlo como una iE.

Referencias

[Kapl09] Kaplan G.N., Doorn J.H., (2009). “Handling Extemporaneous Information in Requirements Engineering”, “Encyclopedia of Information Science & Technology”, IGI Global Books (Mehdi Khosrow-Pour, Information Resources Management Association, USA), Second Ed., Vol 2, pp. 789-794. Total de Páginas 5266. ISBN: 978-1-60566-026-4.

[Kapl14] Kaplan, G.N. (2014). “Información Extemporánea en los Procesos de Requisitos”, Tesis de Maestría, Director Jorge Doorn, Unversidad Nacional de La Matanza (UNLaM), Argentina.

[Kapl15] Kaplan, G., Doorn, J. (2015). “Advanced & Delayed Information in Requirements Engineering” in “Encyclopedia of Information Science & Technology”, IGI Global Books (Mehdi Khosrow-Pour, Information Resources Management Association, USA), Third Ed. Cantidad de Páginas 10384 Vol. 10, ISBN13: 9781466658882, ISBN10: 1466658886, EISBN13: 9781466658899.

[Perl75] Perls F. (1975). "Dentro y fuera del tarro de la basura", Editorial Cuatro Vientos, Chile.

Capítulo

11 Documento de especificación de requisitos

11.1 Tipos de documentos

El documento de especificación [Lapl17] [Leff03] [Wieg13] define el producto software a construir y puede ser descripto desde diferentes perspectivas para asegurar su completa comprensión por todos los que tienen algún interés en él. Según ISO 29148:2018 existen tres documentos principales:

- Especificación de Requisitos de los Involucrados / Stakeholder Requirements Specification (StRS).
- Especificación de Requisitos del Sistema / System Requirements Specification (SyRS).
- Especificación de Requisitos del Software (ERS) / Software Requirements Specification (SRS).

La *Especificación de Requisitos de los Involucrados* describe la motivación de la organización por la cual el sistema se está desarrollando o cambiando, define los procesos, las políticas y las reglas bajo las cuales se utiliza el sistema y documenta los requisitos de alto nivel desde la perspectiva de las partes interesadas, incluidas las necesidades de usuarios/operadores/mantenedores según se derive del contexto de uso.

En un entorno empresarial, la StRS describe cómo la organización busca nuevos negocios o cambia el negocio actual para adaptarse a un nuevo entorno empresarial y cómo utilizar el sistema como medio para contribuir al negocio. La descripción incluye, a nivel de organización; el entorno organizacional, las metas y objetivos, el modelo de negocio, y el entorno de la información, y, a nivel de operación empresarial; el modelo de operación empresarial, modos de operación, calidad operativa del negocio, formación organizacional y concepto del sistema propuesto. Los elementos de información de la StRS deben ser especificados por los interesados. Las partes interesadas deben ser responsable del contenido de la especificación. La StRS sirve como base para la participación activa de los involucrados/as en el proceso de requisitos.

La *Especificación de Requisitos del Sistema* identifica las especificaciones técnicas y de usabilidad para la interacción humano-sistema. Define los requisitos de alto nivel del sistema desde la perspectiva del dominio, junto con información general sobre el *objetivo del sistema*, contexto de uso, restricciones, suposiciones y requisitos no funcionales. Puede incluir modelos conceptuales diseñados para ilustrar el contexto del sistema, las principales entidades del dominio, datos, información y flujos de trabajo. El SyRS presenta una descripción de lo que esperan los clientes del sistema, del entorno esperado del sistema, el perfil de uso del sistema, los parámetros de rendimiento, los calidad y eficacia, y actividades de verificación.

La *Especificación de Requisitos de Software* (ERS) es una especificación para un programa o producto de software en particular. Es importante considerar el papel que desempeña la ERS en el plan total del proyecto. El software puede contener esencialmente toda la

funcionalidad del proyecto o puede ser parte de un sistema más grande. En este último caso normalmente habrá una especificación de requisitos que indicará las interfaces entre el sistema y esta parte del software e impondrá requisitos externos de rendimiento y funcionalidad sobre esa parte del software. Por supuesto, la ERS debería aceptar y ampliar estos requisitos del sistema. También debería indicar la precedencia y criticidad de los requisitos y definir todas las capacidades requeridas del producto software al que aplica, así como documentar las condiciones y limitaciones bajo las cuales el software debe funcionar y los enfoques de verificación previstos para los requisitos.

Los tres documentos de especificación pueden contener elementos de información similares que podrían ser considerados como diferentes vistas del mismo producto, pero son interdependientes. El desarrollo de estos elementos requiere interacción y cooperación, en particular en relación con los procesos de negocios, la práctica organizacional y las opciones para soluciones técnicas. Cada especificación puede tener su propia estructura y organización del documento para asegurar una mejor comprensión.

En lo que sigue del presente capítulo, se describe específicamente todo lo referido al documento de ERS. La importancia de este documento, como ya se mencionó, radica en definir el nuevo sistema de software y asegurar que la realidad que se planifica para cuando el nuevo sistema de software esté en uso, cumpla con las necesidades del cliente-usuario. Representa las bases para un acuerdo entre el cliente y el equipo de desarrollo donde se establece el comportamiento del software. Puede ser utilizada en un contrato legal con el objetivo de hacer cumplir su contenido. Además de la función que cumple con el cliente, el documento de especificación es la guía de todo el proceso de

construcción de software, dando los lineamientos de lo que se debe hacer.

11.2 Características de la ERS

Existen dos maneras de generar una ERS:

1. *Como un modelo autónomo o de forma directa.* Los requisitos son conocidos desde el inicio y pueden ser simplemente enumerados y redactados sin necesidad de mayor interacción o refinamiento.
2. *Como el resultado de un proceso*. Los requisitos emergen a través de actividades como análisis, negociación y validación, en un proceso iterativo de descubrimiento y refinamiento.

En el primer caso, cuando la ERS es autónoma, se requiere de una exhaustiva actividad de verificación y validación (V&V) sobre el documento, con el objetivo de asegurar que los requisitos son correctos, razonablemente completos, sin ambigüedad, consistentes, verificables, modificables y rastreables. Mientras que, en el segundo caso, la actividad de V&V se reduce al mínimo debido a que los requisitos, alojados en modelos previos, han sido verificados y validados oportunamente.

El documento de especificación tiene por objetivo:

- *Definir el alcance del proyecto.* Proporciona una descripción clara de qué debe hacer el software y qué no debe hacer, delimitando el alcance del proyecto.
- *Generar un acuerdo entre las partes.* Sirve como un acuerdo entre el grupo desarrolladores/as y el grupo involucrados/as (principalmente el cliente) sobre lo que se va a construir.

- *Guiar todo el desarrollo.* El equipo de desarrollo utiliza la ERS como base para construir el software de acuerdo con las necesidades especificadas.
- *Probar el software.* El equipo de pruebas utiliza la ERS para verificar que el software cumple con los requisitos establecidos.
- *Controlar los cambios.* La ERS puede ser actualizada para reflejar cambios en los requisitos, manteniendo un control sobre las modificaciones.

Los documentos de especificación son en su mayoría escritos en lenguaje natural, llegando al 93% del total de todos los documentos existentes, y dejando sólo un 7% a otras representaciones [Berr08]. En la Figura 33 se muestra como se distribuye ese 93% según un estudio realizado en la Università di Trento en Italia y disponible en eprints.biblio.unitn.it. A pesar del gran uso del lenguaje natural en este tipo de documentos, cada requisito debe ser representado optimizando su comprensión. Por tal motivo, pueden existir documentos híbridos donde algunos requisitos están expresados mediante una fórmula matemática, un gráfico, seudocódigo o utilizando un lenguaje de especificación.

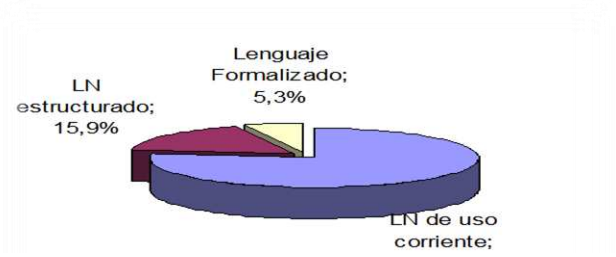


Figura 33 - Uso del LN en ERS

Existen diferentes lenguajes que pueden ser utilizados de manera total o parcial en un documento de requisitos:

- *Lenguajes informales.* Es el lenguaje que utiliza el cliente y los usuarios.
- *Lenguajes semiformales.* No tienen la rigidez de un lenguaje formal, pero proporcionan una estructura más clara que el lenguaje natural. Algunos de ellos son: UML, SysML (Systems Modeling Language), BPMN (Business Process Model and Notation), etc.
- *Lenguajes formales.* Se utilizan para especificaciones matemáticas precisas. Son útiles en sistemas críticos donde la ambigüedad no es aceptable. Tienen el problema de requerir conocimientos avanzados de matemática y lógica. Algunos de ellos son: Z, VDM (Vienna Development Method), Alloy, B-Method, etc.
- *Lenguajes Orientados a Reglas.* Utilizados para definir el comportamiento y las condiciones de un sistema a través de un conjunto de reglas declarativas. Por ejemplo: IBM Operational Decision Manager (IBM ODM).
- *Lenguajes Específicos de Dominio.* Son lenguajes diseñados para un dominio particular. Permiten especificar requisitos de manera más natural y precisa para un área específica. Algunos de ellos son: BPMN (Business Process Model and Notation), SQL (Structured Query Language), etc.

El documento de especificación puede tener diferentes estructuras como ser una lista de sentencias declarativas donde se definen los RF y los RNF; definir cada requisito con algunos atributos para darle más contexto; utilizar un estándar para garantizar las buenas prácticas de especificación (IEEE 830, ISO29148, etc.); entre otras. La forma de abordar un documento de especificación depende de cada proyecto en

particular. En todos los casos, es sustancial remarcar que el acompañamiento de glosarios a los documentos de especificación son de vital importancia ya que ayudan a desambiguar las descripciones.

11.3 Ficha de Requisitos

Una Ficha de Requisitos es una manera más detallada de describir cada requisito, definiendo algunas de sus particularidades. Se pueden seleccionarse los requisitos más complejos de la ERS o especificar todos los requisitos utilizando estas fichas. La ficha permite un control más accesible y rápido a los requisitos, especialmente útil durante el análisis y de gran importancia en el diseño. Algunos factores que se deben tener en cuenta para armar una Ficha de Requisitos:

- *Relevancia para el proyecto.* No todos los atributos son aplicables en cada proyecto. Algunos proyectos pueden requerir un mayor enfoque en la trazabilidad o en la priorización, mientras que otros pueden no necesitar detalles exhaustivos en cada requisito.
- *Enfoque metodológico.* En enfoques *tradicionales* los atributos suelen ser más detallados debido a la necesidad de planificar todo antes de comenzar. En metodologías *ágiles*, como Scrum o Kanban, la especificación de requisitos tiende a ser más ligera, aunque también puede beneficiarse de algunos atributos clave como prioridad y estado.
- *Estándares utilizados.* Si el proyecto sigue un estándar específico, el mismo estándar pueden exigir la incorporación de algunos atributos particulares.

Cabe aclarar que cuando se utiliza una norma no es obligatorio emplear todos los atributos que se sugieren. La clave está en seleccionar los

atributos que agreguen valor a su comprensión y a la gestión de los requisitos. Lo importante es encontrar un equilibrio entre la completitud de los requisitos y la eficiencia en su gestión, adaptándose al contexto del proyecto. En general, los atributos se seleccionan para garantizar que los requisitos puedan gestionarse adecuadamente a lo largo de todo el ciclo de vida del proyecto, desde su construcción inicial hasta su implementación. Los atributos ayudan a proporcionar un contexto adicional, mejorando la comunicación entre equipos técnicos, clientes y otras partes interesadas, al hacer más explícitos los detalles de cada requisito.

Existen algunas ventajas relacionadas con la utilización de atributos:

- *Priorización.* Los atributos como la importancia, urgencia y valor para el negocio permiten identificar cuáles requisitos son prioritarios, ayudando a los equipos a enfocarse en los elementos más críticos para el éxito del proyecto.
- *Trazabilidad.* Los atributos como el origen, la versión y la dependencia facilitan el seguimiento de cada requisito desde su definición hasta su implementación y prueba. Esto asegura que ningún requisito sea pasado por alto y ayuda a mantener un control sobre los cambios.
- *Evaluación de impacto.* Los atributos como el estado (aprobado, en revisión, etc.) y las dependencias entre requisitos permiten analizar el impacto de cambios en un requisito específico sobre otros. Esto es esencial para evaluar el alcance de cambios y minimizar riesgos de re-trabajo.
- *Gestión de riesgos.* Los atributos como la estabilidad y la complejidad ayudan a identificar posibles áreas de riesgo en el proyecto. Los requisitos con alta complejidad o baja estabilidad

pueden requerir una gestión especial o atención adicional para evitar problemas.

- *Planificación y asignación de recursos.* Los atributos como el esfuerzo estimado, la complejidad y el estado permiten al gestor de proyecto planificar mejor el tiempo y los recursos necesarios para cumplir con cada requisito.

Definir los requisitos con sus atributos es fundamental para mejorar la claridad, estructura, trazabilidad y control de la *gestión de los requisitos* a lo largo del ciclo de vida de un proyecto. Los atributos permiten que los equipos de desarrollo, pruebas, y gestión comprendan el contexto completo de cada requisito, ayudando a prevenir errores y malentendidos. En la Figura 34 se presenta un ejemplo de Ficha de Requisitos.

I**D:** 003	
Descripción: El sistema debe identificar y registrar la configuración inicial del ATM.	
Fundamento: Asegurar que los ATM estén configurados para operar según las especificaciones de fábrica.	
Origen: Requerimientos técnicos del equipo.	
Dependencias: La hoja de Excel con la configuración del ATM.	
Tipo: Funcional	**Riesgo**: Medio
Estado: Pendiente	**Complejidad**: Alta
Fecha límite: 15/12/2024	**Prioridad:** Alta

Figura 34 - Ficha de requisitos (Sistema Gestión de ATM)

11.4 Estándares para especificar requisitos del software

Cuando la organización lo solicita o cuando se desea mejorar la estructura y organización del documento, se puede utilizar un estándar. Las ventajas son las siguientes:

- *Mejora la claridad y precisión*. Un documento bien estructurado reduce la posibilidad de malentendidos o interpretaciones incorrectas.
- *Organiza el documento.* Siguiendo una guía se cubran todos los aspectos necesarios para contextualizar y ordenar adecuadamente los requisitos.
- *Mejora el proceso.* Al detallar claramente los requisitos, el estándar ayuda a prevenir errores en la fase de desarrollo y reduce la posibilidad de revisiones costosas más adelante.
- *Asegura el alineamiento.* Los requisitos que siguen un estándar reconocido, como IEEE, ISO o CMMI, ayudan a asegurar que el sistema esté alineado con las regulaciones necesarias, lo que es crucial para certificaciones y auditorías.
- *Mejora la calidad del producto.* Un estándar ayuda a garantizar que los requisitos se recopilen, analicen y especifiquen de manera adecuada.
- *Aumenta la escalabilidad y la reutilización.* Si el proyecto escala, los equipos que se unan tendrán una base clara para trabajar. Además, los requisitos bien especificados pueden ser más fácilmente reutilizados en futuros proyectos o en otros módulos del mismo sistema.

La ISO 29148:2018 define el contenido, la estructura y la organización que debe seguir un documento de especificación de requisitos (ERS). En el Anexo se presenta un ejemplo de una especificación con este estándar. Esta ISO proporciona un marco robusto para la gestión efectiva de requisitos a lo largo del ciclo de vida del software, ayudando a asegurar que el producto final cumple con las expectativas del usuario y los objetivos del proyecto. Algunas particularidades del estándar son las siguientes:

- *Definición de requisitos.* El estándar define una clara clasificación de los diferentes tipos de requisitos que deben abordarse en un proyecto, asegurando que todos los aspectos del sistema estén cubiertos. Propone: requisitos funcionales, requisitos no funcionales**,** requisitos de interfaz, requisitos de diseño y restricciones**.**
- *Estructura y organización.* Proporciona una plantilla recomendada para la creación de un Documento de Especificación de Requisitos del Software (ERS)**.** Esta estructura incluye secciones clave que aseguran que todos los aspectos relevantes de los requisitos sean documentados y cubiertos.
- *Énfasis en la trazabilidad.* Promueve la trazabilidad de los requisitos, es decir, la capacidad de rastrear cada requisito a lo largo del ciclo de vida del sistema.
- *Gestión de cambios.* Aborda específicamente cómo gestionar los cambios en los requisitos. Reconoce que los requisitos son susceptibles de evolucionar durante el ciclo de vida del proyecto.

- *Validación y verificación de requisitos.* Enfatiza la importancia de la verificación y validación de los requisitos. Esto incluye la creación de pruebas, revisiones y análisis formales que aseguren que los requisitos están completos, correctos y viables.
- *Definición de roles y responsabilidades.* Especifica los roles y responsabilidades de las partes interesadas en la Ingeniería de Requisitos.
- *Consideraciones de calidad.* Incluye recomendaciones sobre cómo especificar los atributos de calidad del sistema, como seguridad, rendimiento, fiabilidad y usabilidad. Estos requisitos no funcionales son esenciales para asegurar que el sistema no solo funcione correctamente, sino que también cumpla con los criterios de calidad esperados por los involucrados/as.
- *Adaptabilidad a diferentes tipos de proyectos.* Es suficientemente flexible como para ser aplicada en diferentes tipos de proyectos y metodologías, ya sea un enfoque tradicional o ágil de desarrollo.
- *Enfoque centrado en el usuario.* Presenta un enfoque en las necesidades y expectativas del usuario. El estándar reconoce que los requisitos deben reflejar claramente lo que los usuarios finales esperan del sistema, y promueve la inclusión de modelos previos (casos de uso, escenarios) que representen cómo interactuarán con el software.
- *Gestión de riesgos.* Sugiere identificar los riesgos asociados con los requisitos y los cambios, ya que gestionar estos riesgos desde el principio ayuda a evitar problemas futuros, garantizando que el sistema desarrollado se ajuste a las expectativas iniciales.

Como ya se mencionó, es importante contar con un glosario para desambiguar los requisitos y la norma prevé una sección para este fin en la Introducción ítem "1.3 Definiciones, acrónimos y abreviaturas". De no contar con un glosario se deben incorporar las definiciones directamente en esta sección.

Otros estándares utilizados para especificar requisitos: BABOK (Business Analysis Body of Knowledge), CMMI (Capability Maturity Model Integration), IREB CPRE (International Requirements Engineering Board), IEEE 830:1998 (IEEE Recommended Practice for Software Requirements Specifications), ISO/IEC 25010: 2011 (Software Product Quality Model), ISO/IEC/IEEE 12207:2017 (Software Life Cycle Processes), ISO/IEC/IEEE 15288:2015 (Ciclo de vida de sistemas), ISO/IEC 38500 (Gobernanza de TI), EARS (Easy Approach to Requirements Syntax).

Referencias

[Berr08] Berry, D.M. (2008). "Ambiguity in Natural Language Requirements Documents". In: Paech, B., Martell, C. (eds) Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs. Monterey Workshop 2007. Vol 5320. Springer. https://doi.org/10.1007/978-3-540-89778-1_1

[IEEE830] IEEE Std 830. (1998). "IEEE Recommended Practice for Software Requirements Specifications (ANSI)", IEEE. https://ieeexplore.ieee.org/document/720574

[ISO29148] "Systems and software engineering — Life cycle processes — Requirements engineering". (2018). ISO/IEC/IEEE 29148. https://normasiso.org/norma-iso-29148/#google_vignette

[Lapl17] Laplante, P. A. (2017). "Requirements engineering for software and systems" (3rd ed.). CRC Press.

[Leff03] Leffingwell, D., & Widrig, D. (2003). "Managing software requirements: A unified approach" (2nd ed.). Addison-Wesley.

[Wieg13] Wiegers, K., Beatty, J. (2013) “Software requirements”, (3rd ed.). Microsoft Press.

Capítulo

12 V&V de requisitos

En 1981, Basili encontró cerca de 88 errores en una ERS de 400 páginas para el proyecto "A-7E Operational Flight Program". Esta ERS había sido escrita por un grupo de expertos en especificación de requisitos. En el mismo camino, los estudios de Boehm han demostrado que el impacto potencial de los requisitos mal formulados es sustancial. Boehm estudió como los requisitos, la especificación y los errores de diseño son los más numerosos en un sistema, con un promedio del 64% en comparación con el 36% de los errores de codificación. Ben Achour determinó que la mitad de los casos de uso de un proyecto tenían errores basados en la falta de comprensión de la terminología utilizada. Puede observarse la importancia de una actividad de verificación y validación (V&V) comprometida que permita asegurar que los modelos construidos estén completos, correctos, consistentes, sin ambigüedad y alineados a los objetivos organizacionales.

Cuando se habla de V&V están involucradas las actividades de ***verificación*** y de ***validación*** [Boeh84] [Drus08][Unhe05]. Esta puede ser resuelta en una única actividad o ser realizada en dos actividades por separado. En muchos procesos, especialmente en metodologías ágiles, la verificación y la validación se realizan de manera conjunta y continua en cada iteración o sprint, con el propósito de asegurar que

cada incremento cumple tanto con los estándares como con las expectativas del cliente. Una definición de esta actividad se encuentra en:

> **IEEE 610-1990**, "[V&V es] e*l proceso de determinar si los requisitos para un sistema o componente son completos y correctos, si los productos de cada fase de desarrollo cumplen con las condiciones impuestas por la fase anterior, y si el sistema final o componente final cumple con la ERS"*.

En enfoques tradicionales, estas actividades suelen realizarse por separado y la verificación antes que la validación. De esta manera se pueden identificar y corregir problemas en los requisitos desde muy temprano en el proceso de construcción del software y antes de ser evaluados por el cliente-usuario. Detectar problemas tempranamente evita costosos errores en etapas posteriores reduciendo los riesgos de fallos en el proyecto. Para diferenciar la verificación de la validación se pueden responder los siguientes interrogantes. Cuando se *verifica* se debe responder ¿estamos construyendo el producto correctamente? y cuando se *valida* se debe responder *¿estamos construyendo el producto correcto*?

El aseguramiento de la calidad (QA) está estrechamente relacionado con la verificación y validación (V&V) de requisitos, ya que ambos procesos tienen como objetivo garantizar que el software cumple con los estándares de calidad y con las expectativas del cliente. QA abarca prácticas, procesos y técnicas que aseguran que los requisitos sean adecuados, claros y que estén alineados con los objetivos del proyecto, mientras que V&V son métodos específicos dentro de QA que aseguran que el software cumpla con los requisitos correctos y se desarrolle de manera efectiva.

12.1 Verificación de requisitos

El objetivo de la verificación es asegurar que todos los requisitos construidos estén correctos y completos. Cabe destacar que el proceso de definir requisitos es inherentemente incompleto porque intenta sintetizar el mundo real, el cual está en constante movimiento. De esta manera, se debe procurar tener requisitos los más completos posibles.

Para verificar los requisitos se utilizan diferentes técnicas, las cuales pueden ser clasificadas en estáticas, dinámicas y formales [Hall90] [Somm11].

Las *técnicas estáticas* analizan la forma y la estructura de un artefacto. Estas técnicas no implican la ejecución del sistema, sino que se basan en revisiones y análisis para encontrar errores o inconsistencias en los requisitos. Entre las más utilizadas se encuentran:

- *Inspección de requisitos.* La inspección es una técnica formal de revisión que sigue un proceso estructurado y generalmente incluye roles específicos (ver sección 12.1.1). Este proceso es detallado y meticuloso, enfocado en identificar errores, inconsistencias y ambigüedades en el documento de especificación. Las inspecciones suelen ser muy efectivas en la detección de defectos y se documentan exhaustivamente, lo cual es crucial en proyectos de alta criticidad donde se necesita precisión y rastreabilidad [Wieg13] [Somm11].
- *Revisiones de requisitos.* Una revisión de requisitos es una técnica más general y flexible. Puede ser formal o informal y suele involucrar a un grupo más amplio de partes interesadas. En las revisiones, los participantes revisan el documento de requisitos de manera global, a menudo sin roles específicos o un

proceso tan estricto como en la inspección. El objetivo de la revisión es asegurar que los requisitos estén alineados con las expectativas de los usuarios y del negocio y que no existan problemas graves [Pres20].

Las técnicas dinámicas suelen involucrar algún nivel de ejecución como una simulación o prototipado. Estas técnicas permiten verificar y validar los requisitos al interactuar o simular sus funcionalidades.

Las técnicas formales se basan en el uso de notaciones matemáticas para especificar, analizar y verificar los requisitos. Estas técnicas son más rigurosas y suelen aplicarse en proyectos de alta criticidad y utilizan algún lenguaje formal:

- *Lenguaje de especificación Z.* Es un lenguaje formal basado en la teoría de conjuntos y la lógica de primer orden. Z permite definir el estado de un sistema mediante esquemas que contienen variables de estado y sus restricciones. La estructura modular de Z permite dividir la especificación en componentes más pequeños y reutilizables. Este lenguaje es ampliamente utilizado en sistemas críticos, como sistemas financieros y de control de tráfico, donde los errores pueden tener consecuencias graves.
- *VDM* (Vienna Development Method). Es otro enfoque formal que utiliza una combinación de modelos algebraicos y de invariantes para especificar y verificar requisitos. VDM define el sistema a través de especificaciones abstractas y permite una refinación progresiva del sistema. Además, utiliza invariantes para garantizar que ciertas condiciones se mantengan siempre verdaderas en el estado del sistema.

Cabe destacar la importancia de automatizar el proceso de verificación de requisitos, haciendo posible una *verificación continua* y a bajo costo. La implementación de la automatización en enfoques tradicionales puede ser compleja y requiere un esfuerzo inicial significativo en la configuración de herramientas y el diseño de casos de prueba automáticos. Sin embargo, cuando se realiza correctamente, aporta mejoras significativas en la calidad y confiabilidad del software.

12.1.1 Inspecciones de software

Las inspecciones se han utilizado ampliamente en la Ingeniería de Software para verificar y validar, primero programas y luego, otros documentos creados durante el proceso de desarrollo del software [Faga76] [Basi94]. El objetivo es detectar Discrepancias, Errores y Omisiones (DEO) en los requisitos como también asegurar que se ha cumplido con los estándares de calidad de la organización. Las inspecciones pueden ser aplicadas a diferentes tipos de artefactos desde etapas tempranas, permitiendo el desarrollo de un producto software con menos probabilidad de fracaso. El proceso de inspección de Fagan tiene ciertas particularidades como roles, materiales y fases que se describen a continuación.

Roles:

- *Director.* Planifica la inspección, asigna recursos y chequea resultados.
- *Moderador.* Dirige la actividad de inspección y facilita la interacción entre los miembros del equipo. Asegura que las inspecciones sean efectivas y eficientes.
- *Productor.* Crea los materiales a ser inspeccionados y se encarga de las correcciones.

- *Inspector.* Escribe cada defecto anotando la categoría, severidad, tipo y origen del mismo.
- *Revisor.* Eleva los problemas y consideraciones acerca del producto sin proponer soluciones.
- *Lector.* Lee partes del producto para enfocar la atención en un punto problemático particular.

Materiales (recursos necesarios para realizar una inspección):

- *Notificación de la reunión de inspección.* Tiene como objetivo informar al equipo de inspectores de la inspección por realizar.
- *Lista de defectos de la inspección.* Es usado por los inspectores para registrar los defectos.
- *Resumen de defectos de la inspección.* Lo elabora el inspector y realiza después de la reunión. Resume el tipo, clase y severidad de todos los defectos acordados durante la reunión.
- *Reporte administrativo de la inspección.* Lo elabora el moderador al final de la etapa de reunión de la inspección con distintas métricas de todo el proceso.

Fases del proceso de inspección:

- *Fase de planeamiento* (planning). Consiste en la selección del material a inspeccionar, la elección de los participantes, la determinación de los roles (inspector, moderador y escriba) y la preparación del material a inspeccionar.
- *Fase de descripción general* (overview). Consiste en hacer conocer a los participantes acerca de los materiales a ser examinados y asignar roles.
- *Fase de preparación* (preparation). Es realizada por el inspector una vez que recibe el material y una copia del plan de

inspección. La preparación consiste primero en la lectura cuidadosa de las instrucciones y luego en completar los formularios de inspección, registrando toda DEO detectada.

- *Fase de reunión* (meeting). Apunta principalmente a confirmar o rechazar las DEO detectadas y secundariamente a descubrir nuevas DEO. En la reunión participan un moderador, un escriba, el inspector y los autores. Los autores convocados a la reunión realizan posteriormente las correcciones necesarias.
- *Fase de corrección* (rework). Los autores corrigen los errores consensuados en la reunión.
- *Fase de seguimiento* (follow up). Se inspecciona que todos los defectos encontrados en la reunión de la inspección efectivamente se hayan corregido y que no hayan aparecido nuevos.

En la Figura 35 se puede observar que el proceso comienza planificando la inspección (Planeamiento). Luego, los inspectores se familiarizan con todo el material a inspeccionar y se define de qué forma se debe llevar adelante el trabajo (Descripción General y Preparación). Una vez finalizada estas fases se realiza la Reunión donde los inspectores y autores deciden qué modificaciones deben ser impactadas en el artefacto original (Reunión).

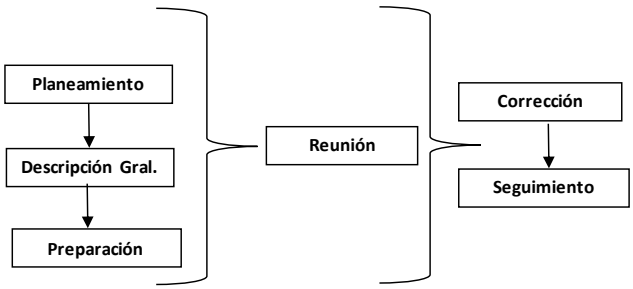


Figura 35 - Proceso de Inspección de Fagan

A partir de las DEOs detectadas durante la Preparación y confirmadas en la Reunión, los autores corrigen el modelo (Corrección).

Cabe destacar que la corrección del artefacto original siempre está a cargo de los autores, de la inspección solo se obtiene una lista de recomendaciones. Para finalizar, se corrobora que las correcciones fueron realizadas correctamente (Seguimiento). Cuando las DEOs son muchas o su importancia es alta, se pueden realizar nuevas inspecciones sobre los mismos artefactos.

Mediante el uso de las inspecciones el número de errores en el producto final puede disminuir significativamente creando un producto de mayor calidad. Adicionalmente, en el futuro, el equipo de diseño y desarrollo será capaz de evitar esos errores frecuentes detectados en las inspecciones. A las mejoras de calidad, cabe agregar la citada disminución de costos derivados de la prevención y detección temprana de errores.

El beneficio de las inspecciones lo demuestra O'Neill en su *Experimento Nacional de Calidad de Software* (NSQE) en 1991. Aquí se recopilaron datos de defectos y prácticas de inspección de docenas de compañías organizadas por niveles de madurez, tipo de organización, tipo de producto, lenguaje de programación y región global. Como resultado las compañías que usaban inspecciones obtenían como ganancia de 4 a 8 dólares por cada dólar invertido. Se comprobó, además, que un defecto que pasa a la próxima fase puede costar diez veces más detectarlo y corregirlo, con lo que se corrobora que cuando un defecto se filtra en la prueba, por ejemplo, se deben ejecutar múltiples pruebas para confirmarlo y reunir información acerca de él. Esto consume tiempo y se aparta del hilo principal de trabajo.

12.1.2 Verificación cuantitativa de requisitos utilizando GQM

Una forma de determinar la calidad es realizando una verificación que permita medir cada requisito en particular tomando en cuenta las *características* que lo definen. A tal efecto se utiliza la clasificación de Wiegers [Wieg99] que establece las siguientes características de los requisitos: necesario, correcto, conciso, completo, consistente, no ambiguo, verificable, trazable y modificable. A estas características se les incorporaron tres: autónomo (no necesita de otro requisito para ser comprendido), singular (describe un único requisito) y punto de vista (en la ERS se recomienda que sea del software).

Para determinar el grado de cumplimiento de cada requisito se utiliza una métrica cuantitativa obtenida con el enfoque de Basili: *Goal Question Metric* (GQM) [Basi92]. Cabe destacar que este enfoque es útil para trabajar con cada requisito en particular, pero para liberar la ERS se debe evaluar también el conjunto de requisitos para asegurar que son consistentes y que no existen contradicciones o conflictos entre ellos.

El modelo GQM tiene tres niveles:

1. Nivel conceptual (OBJETIVO / GOAL)

   Objetivo: determinar la calidad de cada requisito de software.

2. Nivel operativo (PREGUNTA / QUESTIONS)

   Tipo de Respuesta: SI/NO

   Pregunta y **Peso** de cada característica:

Características	Preguntas	Peso en la fórmula
Correcto	**(P1)** ¿está dentro del alcance del *objetivo del sistema*?	NO=1 / SI=0
Conciso	**(P2)** ¿Es de fácil comprensión? (se puede comprender en una primera lectura)	NO=1 / SI=0
Necesario	**(P3)** ¿Puede ser reemplazado por otra funcionalidad?	SI=1 / NO=0
Verificable	**(P4)** ¿Están los parámetros/valores que permiten aplicar un método cuantificable?	NO=1 / SI=0
Completo	**(P5)** ¿Están los datos necesarios para implementar el requisito?	NO=1 / SI=0
Consistente	**(P6)** ¿Existe alguna contradicción en el propio requisito?	SI=1 / NO=0
Autónomo	**(P7)** ¿Alcanza con analizar este requisito solo? (sin conocer otros requisitos)	NO=1 / SI=0
Indivisible	**(P8)** ¿La realización del requisito corresponde a una única funcionalidad del software?	NO=1 / SI=0
Punto de Vista	**(P9)** ¿Esta escrito desde la perspectiva correcta?	NO=1 / SI=0
No ambiguo	**(P10)** ¿Puede tener más de una interpretación?	SI=1 / NO=0
Trazable	**(P11)** ¿Se comprende el origen del requisito?	NO=1 / SI=0
Modificable	**(P12)** ¿Es posible de ser modificado forma fácil, completa y consistente?	NO=1 / SI=0

Tabla 10 - Preguntas del Nivel Operativo de GQM

3. Nivel cuantitativo (MÉTRICA / METRIC)

Fórmula: 0 <= (∑ preguntas / Cantidad de preguntas) <= 1

Esta fórmula se utiliza para evaluar cada característica de un requisito y luego, se utiliza para evaluar todo el requisito. Para cada característica el valor puede ser 0 o 1 (Rango x Característica). Luego, se analiza cada requisito en particular, si el valor está entre 0.1 y 0.59 se envía a

revisión (Rango x Requisito), si es menor a ese rango es aceptado y si es mayor es rechazado. "Aceptado" significa que tiene la calidad suficiente para pasar a la ERS. "A revisión" significa que debe ser corregido y pasar por otra verificación. "Rechazado" es cuando un requisito no es correcto, o sea, no pertenece al dominio en estudio.

Evaluación:

Rangos x CARACTERÍSTICA	
Aceptado	Rechazado
0	1

Rangos x REQUISITO		
Aceptado	A revisión	Rechazado
0	>=0.1 y <=0.59	>=0.6

Tabla 11 - Evaluación de cada requisito

Es importante remarcar que los valores de la tabla *Rangos x Requisito* pueden ser ajustados para contextos más críticos o particulares.

Se puede observar que algunas características tienen una relación muy estrecha entre sí, por ejemplo, cuando un requisito no está completo genera ambigüedad y a su vez hace dificultosa la trazabilidad. Por tal motivo, se sugiere responder cada pregunta de manera independiente. A modo de ejemplo, en la Figura 36 se muestra la evaluación de características realizada sobre al REQ-009 de la ERS del Anexo.

Se puede observar que en la Figura 36 la columna "Peso" contiene 0 o 1 de acuerdo a la definido en la Tabla 10. Se suma la columna "Peso" para obtener el "Total x Requisito", este valor indica la calidad del mismo. Este total es utilizado luego para calcular la columna "Fórmula" de la Tabla 12. La columna "Observaciones" describe las dudas que aparecen durante la evaluación del requisito. Esta columna puede convertirse en un mecanismo de corrección valioso, ya que puede ser entregado a los autores para que se enfoquen directamente en cómo mejorar cada descripción. Una vez corregidos, la decisión de

volver a verificar esos requisitos o directamente aceptarlos es responsabilidad del ingeniero de requisitos.

ID_REQUISITO: REQ-009

Descripción: El sistema debe permitir liberar las OC reservadas previamente.

Característica	Peso	Observaciones
Correcto	1	¿Puede liberar cualquier persona o debe estar autorizada?
Conciso	0	
Necesario	0	
Verificable	1	Falta información
Completo	1	¿A qué reserva se refiere?¿es por cliente o se reserva en general?
Consistente	0	
Singular	0	
Autónomo	0	
Punto de Vista	0	
No ambiguo	1	¿La reserva es cuando ya se realizó el pedido o antes?
Trazable	1	No se conoce el contexto de la reserva.
Modificable	1	Falta información
TOTAL x Requisito	**6**	

Figura 36 - Ejemplo de análisis de cada requisito

Id. Requisito	Fórmula *Total x Requisito / Cantidad de requisitos*	Decisión por requisito *Aceptado = 0 A revisión >= 0.1 y <= 0.49 Rechazado >= 0.5*
REQ-001	0 / 10 = 0	Aceptado
REQ-002	0 / 10 = 0	Aceptado
REQ-003	2 / 10 = 0,2	A revisión
REQ-004	0 / 10 = 0	A revisión
REQ-005	6 / 10 = 0,1	A revisión
REQ-006	5 / 10 = 0,3	A revisión
REQ-007	0 / 10 = 0	Aceptado
REQ-008	0 / 10 = 0	Aceptado
REQ-009	6 / 10 = 0,6	**Rechazado**

Tabla 12 - Ejemplo de análisis de toda la ERS

Finalmente, una vez corregido el requisito REQ-009 se describió de la siguiente manera: “El sistema debe permitir al responsable de compras liberar números de OC reservados previamente.”.
Esta verificación puede ser implementada en diferentes tipos proyectos haciendo los cambios que se consideren necesarios.

12.2 Validación de requisitos

La validación tiene por objetivo asegurar que los requisitos reflejan las necesidades y expectativas de los involucrados/as, que estén alineados con los objetivos organizacionales y que capturan fielmente lo que se espera del sistema. En ambientes de muchos cambios, los requisitos pueden ser monitoreados por una validación automatizada. El monitoreo continuo de los sistemas en producción proporciona datos en tiempo real sobre el uso del sistema y posibles problemas. Esto ayuda a identificar requisitos no anticipados o cambios necesarios en los requisitos actuales. De esta manera, los requisitos pueden ser validados continuamente en función de datos reales de producción. Si un requisito no está cumpliendo con las expectativas de rendimiento o funcionalidad, rápidamente se pueden realizar los ajustes necesarios.
Existen diferentes técnicas para validar requisitos:

- *Prototipos o simulaciones.* Se crea un modelo funcional del sistema para validar los requisitos con los usuarios.
- *Revisiones con involucrados/as.* Se realizan reuniones y sesiones de trabajo con todos los participantes.
- *Pruebas de aceptación.* Se crean pruebas para verificar que el software final cumpla con los requisitos especificados.

12.2.1 Prototipos

El uso de prototipos permite a los involucrados/as interactuar con una representación funcional del sistema antes de comenzar con la construcción del mismo. La creación de prototipos se emplea comúnmente para validar la interpretación de los requisitos existentes, aclarar los atributos de los requisitos e identificar requisitos omitidos (requisitos tardíos). Facilitan la interpretación de los supuestos generados durante la construcción de los requisitos y pueden proporcionar información útil sobre por qué están equivocados. Esta técnica se utiliza una vez que se tiene una ERS parcial o completa., para obtener una retroalimentación temprana de posibles errores y omisiones, lo que minimiza el riesgo de aumento del costo del software en etapas posteriores. Existen diferentes tipos de prototipos:

- *Prototipos de baja fidelidad.* Son representaciones simples y rápidas del sistema, a menudo en papel o mediante herramientas básicas. Se centran más en la estructura y la interacción que en los detalles.
- *Prototipos de alta fidelidad.* Representan una simulación más cercana del sistema final, con más detalles en la interfaz de usuario, interacciones y funcionalidad.

Elegir la forma de construir el prototipo dependerá de las necesidades del proyecto. Algunas de las desventajas pueden ser disminuidas sensiblemente al utilizar prototipos de baja fidelidad ya que se alejan de cuestiones de diseño que pueden traer tensión cuando el software esté terminado y no coincida la versión final con la propuesta de validación. Para ello, se debe tener en cuenta que cuanto menos cosmética tenga el prototipo mejor será interpretado por los usuarios y bajarán las

expectativas irreales. Durante la ejecución del prototipo se pueden crear espacios (pantallas, inputs, etc.) adicionales donde se pueda ingresar toda aquella información que aparezca espontáneamente durante la validación y de esta manera, dejarla empotrada en el mismo prototipo como documentación de respaldo.

En la Tabla 13 se muestran las ventajas y desventajas del uso de prototipos.

VENTAJAS	DESVENTAJAS
Clarificación de requisitos: permiten a los involucrados/as visualizar y experimentar cómo funcionará el sistema, reduciendo la ambigüedad en la interpretación de los requisitos.	Expectativas irreales: los usuarios pueden malinterpretar el prototipo como una versión casi final del sistema y desarrollar expectativas poco realistas sobre los plazos o las funcionalidades.
Detección de requisitos tardíos**:** al interactuar con el prototipo, los usuarios pueden identificar problemas, malentendidos y requisitos omitidos antes de que el sistema esté completamente desarrollado.	Costos de desarrollo: aunque los prototipos desechables son rápidos y baratos de producir, los prototipos de alta fidelidad o evolutivos pueden requerir una inversión considerable en términos de tiempo y recursos.
Alineación con los involucrados/as: ayuda a garantizar que el sistema final cumpla con las expectativas y necesidades de los usuarios al participarlos activamente durante el desarrollo.	Poco centrado en los RNF**:** a menudo, los prototipos se centran en aspectos funcionales e interfaces, y es posible que no capten de manera adecuada los requisitos no funcionales (como el rendimiento o la seguridad).
Adaptabilidad: pueden modificarse fácilmente en respuesta a la retroalimentación, permitiendo iteraciones rápidas y ajustes constantes a los requisitos.	Desvío del proceso formal: en algunos casos, los equipos pueden confiar tanto en los prototipos que se descuida la documentación formal de requisitos, lo que podría causar problemas en la fase de implementación.

Tabla 13 - Ventajas y desventajas del uso de prototipos de validación

Existen diferentes herramientas para crear prototipos:

- *Balsamiq.* Ideal para prototipos de baja fidelidad.

- *Figma.* Permite crear prototipos interactivos de alta fidelidad y colaborar en tiempo real con los involucrados/as.
- *Adobe XD*. Herramienta para diseñar prototipos de interfaces interactivas con funciones avanzadas para la validación de UX.

Referencias

[Basi92] V.R. Basili (1992). "Software Modeling and Measurement: The Goal Question Metric Paradigm,", Computer Science Technical Report Series, CS-TR-2956 (UMIACS-TR-92-96), University of Maryland, College Park, MD.

[Basi94] Barnard, J., Price, A. (1994). "Managing Code Inspection Information". IEEE Software. pp 59-69.

[Boeh84] Boehm, B. W. (1984). "Verifying and validating software requirements and design specifications". *IEEE Software, 1*(1), 75-88.

[Drus08] Drusinsky, D., Michael, J. B., & Shing, M. T. (2008). "A visual tradeoff space for formal verification and validation techniques". IEEE Systems Journal, 2(4), 513-519.

[Faga76] Fagan, M.E., (1976). "Design and Code Inspections to reduce Errors in Program Development", IBM Systems Journal, Vol.15, Nº3, pp.182-211.

[Hall90] Hall, A. (1990). "Seven myths of formal methods". *IEEE Software, 7*(5), 11-19.

[IEEE610] IEEE Std 610-1990, "IEEE Standard Glossary of Software Engineering Terminology", IEEE.

[Pres20] Pressman, R. S., Maxim, B. R. (2020). "Software engineering: A practitioner's approach" (9th ed.). McGraw-Hill Education.

[Somm11] Sommerville I. (2011). "Ingeniería de Software", Edición 9, Pearson,.

[Unhe05] Unhelkar B., (2005). "Verification and validation for quality of UML 2.0 models", vol. 42. John Wiley & Sons.

[Wieg13] Wiegers, K., Beatty, J. (2013). “Software requirements”, (3rd ed.). Microsoft Press.

[Wieg99] Wiegers K. (1999). “Writing Quality Requirements”. Disponible en: http://www.processimpact.com/articles/qualreqs.pdf

Capítulo

13 Gestión de requisitos

El propósito de la gestión de requisitos es garantizar que los objetivos del proyecto se cumplan con éxito. Es un conjunto de técnicas para documentar, analizar, priorizar y rastrear los requisitos. De esta manera el equipo de desarrollo puede incorporar los cambios en los requisitos de manera rápida y segura. La gestión de requisitos proporciona una forma de evitar errores inesperados. A pesar de ser una actividad propia de la Ingeniería de Requisitos es transversal a todo el ciclo de vida del software como se puede observar en la Figura 37, siendo clave para asegurar que los requisitos se mantengan consistentes, trazables y alineados hasta el producto final. También se debe tener en cuenta la creciente complejidad de los productos tecnológicos conectados (dispositivos IoT, vehículos inteligentes, sistemas de automatización del hogar, plataformas industriales, etc.) que hace que la gestión de requisitos sea crucial ya que permite controlar la interoperabilidad, seguridad, actualizaciones y cambios, asegurando que se cumplan con todos los objetivos, expectativas y regulaciones previstas.

El gráfico de la Figura 37 enfatiza como la Ingeniería de Requisitos no es una actividad aislada, sino que se integra a lo largo de todo el ciclo de vida del software. Su correcta ejecución influye en la calidad del

diseño, el desarrollo, las pruebas y la sostenibilidad del sistema en producción.

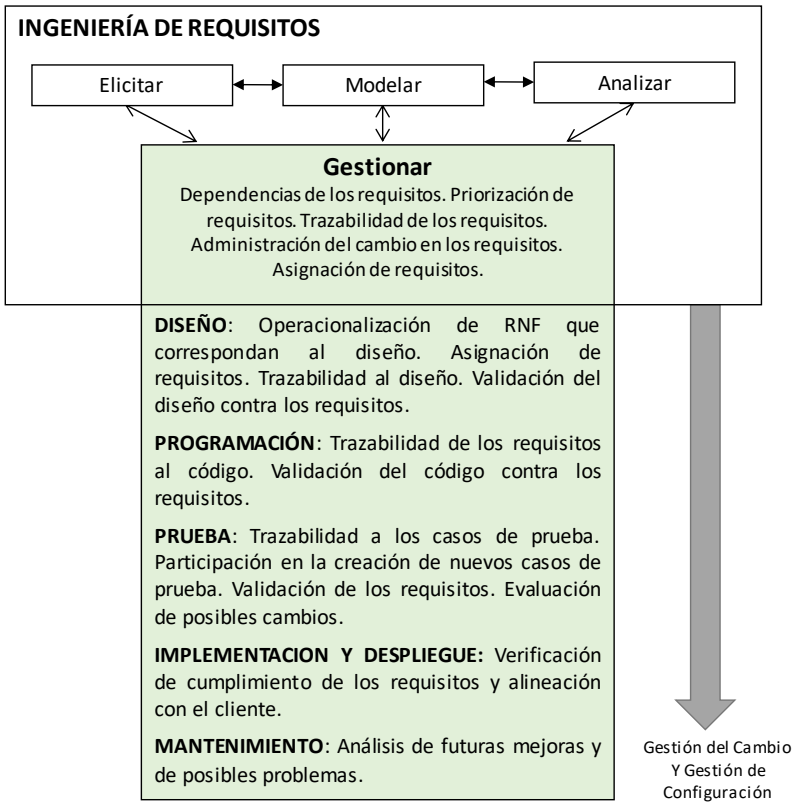


Figura 37 - Gestión de Requisitos

Se puede observar en la Figura 37 que el objetivo de la gestión de requisitos es asegurar, en cualquier momento de la construcción del software, que los requisitos estén actualizados, trazables y alineados con los objetivos del proyecto. También es importante destacar que a partir de la primera ERS que se libera (parcial o total) la Ingeniería de Requisitos, particularmente la gestión de requisitos, comienzan a interactuar con la Gestión del Cambio y la Gestión de Configuración asegurando que en todo momento se cumple con los requisitos definidos con el cliente y se mantiene actualizado el modelo de trazabilidad.

IBM define la gestión de requisitos de la siguiente manera:

> **IBM [15] ,** *"Con la gestión de requisitos, puede superar la complejidad y las interdependencias que existen en los ciclos de vida de la ingeniería actuales para agilizar el desarrollo de los productos y acelerar el despliegue. Los problemas en la gestión de requisitos se citan a menudo como las principales causas de los fracasos de los proyectos. Tener requisitos inadecuadamente definidos puede resultar en un aumento del alcance, retrasos en los proyectos, sobrecostes y una mala calidad del producto que no cumple con las necesidades del cliente y los requisitos de seguridad. Disponer de un plan de gestión de requisitos es fundamental para el éxito de un proyecto, ya que permite a los equipos de ingeniería controlar el alcance y dirigir el ciclo de vida del desarrollo del producto. El software de gestión de requisitos proporciona las herramientas para ejecutar ese plan, lo que ayuda a reducir costes, acelerar el tiempo de comercialización y mejorar el control de calidad".*

El Modelo de Madurez de Capacidades Integrado (CMMI) [CMMI06] determina en el Nivel 2, conformado por 7 áreas de proceso que contribuyen a proyectar la eficacia de la gestión, que la gestión de requisitos se encarga de obtener, comprender, aprobar los requisitos, así como gestionar los cambios y mantener la trazabilidad bidireccional identificando las inconsistencias entre el trabajo real que se va a llevar a cabo y los requisitos. Básicamente identificar inconsistencias entre los requisitos y los planes de proyecto.

[15] https://www.ibm.com/es-es/topics/what-is-requirements-management#:~:text=Disponer%20de%20un%20plan%20de,vida%20del%20desarrollo%20del%20producto.

La gestión de requisitos, según ISO 29148:2018, abarca varias actividades esenciales para garantizar que los requisitos se manejen de manera efectiva a lo largo del ciclo de vida del sistema. Las ideas clave que aborda el estándar son: trazabilidad, priorización, gestión del cambio, control de la configuración, validación y documentación.

Para gestionar con seguridad los requisitos es necesario contar con algún tipo de automatización que reduzca los errores en todo el proceso, incremente la colaboración entre el equipo de desarrollo, facilite la incorporación de los cambios, maneje las versiones de forma segura y permita que los requisitos lleguen al software lo más seguro y rápido posible. Existen diversas herramientas y aplicaciones especializadas para la *gestión de requisitos*, que ayudan a los equipos de desarrollo a definir, priorizar, rastrear y gestionar los requisitos de manera eficiente. Estas herramientas permiten una mejor colaboración, visibilidad y control sobre los requisitos a lo largo del ciclo de vida del proyecto. Algunas de ellas son:

- *IBM DOORS*. Permite la gestión de requisitos jerárquica y la trazabilidad completa; análisis de impacto y soporte para requisitos normativos; gestión colaborativa y visualización del ciclo de vida de los requisitos; soporte para metodologías ágiles, híbridas y tradicionales.
- *Azure DevOps***.** Permite la gestión de tareas, historias de usuario y requisitos; seguimiento de cambios, dependencias y relaciones entre requisitos; integración con herramientas de CI/CD y de prueba; informes y análisis en tiempo real sobre el progreso de los requisitos.
- *Siemens Polarion***.** Permite la gestión colaborativa y centralizada de requisitos; trazabilidad de requisitos a lo largo

del ciclo de vida del producto; soporte para revisiones y aprobaciones electrónicas; compatible con modelos ágiles, en cascada e híbridos.

- *Jira* (con el complemento Jira Software y Jira Align). Permite la creación y priorización de historias de usuario, tareas y requisitos; visualización de proyectos con tableros Kanban y Scrum; seguimiento de dependencias y cambios en los requisitos; integración con otras herramientas como Confluence y Bitbucket.
- *ReqView*. Permite la recolección y organización de requisitos en una interfaz fácil de usar; trazabilidad de requisitos desde el diseño hasta la implementación y pruebas; edición colaborativa; exportación e importación de documentos de requisitos.

La elección de la herramienta adecuada dependerá del tipo de proyecto, de su complejidad, del tamaño del sistema, de la metodología elegida, etc. Se debe tener en cuenta que la integración de herramientas de IA en la gestión de requisitos puede permitir una mayor eficiencia, reducción de errores, y mejor colaboración. Estas herramientas pueden ajustarse según el tamaño y la complejidad del proyecto, ofreciendo capacidades como análisis predictivo, trazabilidad automatizada, y generación de informes. Otros autores proponen el uso de ontologías [Assa10] [Farf11] [Cast14] [Parr15] [Avde16] [Sitt17] [Trok18] para mejorar la especificación de los requisitos y de esta manera, optimizar la trazabilidad y la gestión del cambio. Se debe tener en cuenta que el uso de ontologías implica desafíos relacionados con su diseño, implementación y mantenimiento, además de requerir experiencia técnica y del dominio para su correcta aplicación.

Entre las sub-actividades más importantes que encierra la gestión de requisitos se encuentran:

- Dependencias de los requisitos.
- Priorización de requisitos.
- Trazabilidad de los requisitos.
- Administración del cambio.
- Asignación de requisitos.

13.1 Dependencias de los requisitos

Se refiere a la estrecha relación que tienen algunos requisitos entre sí. Esta relación indica que un requisito está condicionado por, o influye en, uno o más requisitos, lo que puede afectar cómo se gestionan. Las dependencias ayudan al gestor de proyecto a organizar el orden de implementación. Esto permite que el equipo de desarrollo trabaje de manera eficiente, evitando bloqueos por la falta de funcionalidades base. También ayudan a identificar posibles conflictos entre requisitos que puedan interferir entre sí. Esto facilita la resolución temprana de problemas antes de que impacten negativamente en el software.

Existen diferentes tipos de dependencias:

- *Dependencia funcional.* Un requisito depende de otro para poder funcionar.
- *Dependencia temporal.* Un requisito no se puede implementar hasta que otro esté completado.
- *Dependencias transitivas.* Un requisito A depende de un requisito B, y este último, a su vez, depende de un requisito C. Así, A se ve afectado indirectamente por C.
- *Dependencia de recursos.* Un requisito puede depender de la disponibilidad de ciertos recursos.

Al analizar cada ERS pueden aparecer nuevos tipos de dependencias, lo que lleva a determinar que las dependencias pueden ser particulares en cada proyecto. En desarrollos con muchos requisitos definir las dependencias puede ser un trabajo muy engorroso y difícil de mantener en el tiempo, esto se relaciona con la cantidad de cambios del contexto. Para evitar errores y tiempos de actualización, se debe elegir alguna forma de automatización. Entre las herramientas más utilizadas están los *diagramas de dependencia* que muestran visualmente cómo están conectados los requisitos. También existen otros como los *diagramas de árbol*, los *diagramas de causa-efecto* o *grafos*. Se pueden documentar en las *matrices de trazabilidad* ya que éstas muestran cómo un requisito afecta o está afectado por otros requisitos del sistema o incluir en la ERS alguna sección de dependencias donde se puedan enumerar los identificadores de los requisitos relacionados y explicar la naturaleza de la dependencia. A continuación, se describe un ejemplo de dependencias entre requisitos para la funcionalidad que permite reservar números de OC, pero solo por personal autorizado.

Las dependencias son necesarias para *priorizar* correctamente los requisitos. Aquellos que son fundamentales o que tienen dependencias críticas, se implementan antes o en paralelo a los que dependen de ellos.

Para la *trazabilidad* de requisitos las dependencias son fundamentales. Esto implica que cualquier cambio o ajuste en un requisito puede rastrearse y se puede ver qué otros requisitos están afectados, ayudando a garantizar la consistencia del sistema. Las dependencias permiten evaluar el impacto del cambio en otros requisitos. Esto es clave en la gestión del cambio para garantizar que no se rompa la coherencia del sistema cuando cambia un requisito.

La Figura 38 es un ejemplo relacionado con la ERS del Sistema de Gestión de ATM, donde se puede observar las dependencias y relaciones generadas entre los requisitos.

Nivel 1: Requisito principal
[REQ-010: El sistema debe asegurar que solo personas autorizadas realicen la reserva de números de OC]
Nivel 2: Requisitos dependientes principales
1. [REQ-011: El sistema debe registrar los usuarios autorizados para realizar las reservas]
2. [REQ-012: El sistema debe verificar que la reserva tenga todas las autorizaciones antes de proceder a la reserva]

Nivel 3: Requisitos dependientes secundarios
1. [REQ-013: El sistema debe garantizar que solo usuarios autorizados hagan la reserva]
2. [REQ-007: El sistema debe registrar el responsable, fecha, hora y números de OC reservados]

Nivel 4: Requisitos adicionales de validación y seguimiento
1. [REQ-014: El sistema debe auditar las reservas de números de OC]
2. [REQ-015: El sistema debe notificar intentos de reserva no autorizados]

Figura 38 – Dependencias y relaciones de los requisitos

13.2 Priorización de requisitos

La tarea de asignar prioridades requiere de la participación del cliente-usuario con cierto nivel de decisión y puede realizarse de diversas maneras, tales como reuniones, cuestionarios y otras. Se debe determinar la importancia relativa que tiene un requisito para el cliente-usuario, y organizar aquellos requisitos que deben implementarse inicialmente frente a aquellos que pueden postergarse. Al asignar prioridades, se deben tener en cuenta la dependencia entre requisitos, la multiplicidad de intereses del cliente-usuario, las limitaciones de recursos, las necesidades del negocio, las imposiciones del mercado y

los costos de implementación, entre otros factores. Por ende, el ingeniero de requisitos debe orientar al cliente-usuario respecto a estas contingencias.

La asignación de prioridades ha sido muy estudiada por una variedad de autores que han definido estrategias simples y complejas, que van desde una asignación que califica al requisito como importante o no, como obligatorio o diferible, hasta un ranking de importancia. Dentro de estas técnicas de priorización se encuentran:

- *Asignación Numérica.* Que divide los requisitos en tres grupos: obligatorios, deseables y no esenciales [Brac90].
- *Ranking.* Se basa en una escala ordinal donde al requisito más importante se le da el valor 1 y al menos importante el valor N siendo N la cantidad de requisitos, y la lista de ranking se obtiene aplicando por ejemplo el método de ordenamiento burbujeo o árbol binario de búsqueda.
- *AHP* (Analytic Hierarchy Process). Basada en la técnica de "pairwise comparison" estimando un valor relativo de importancia de los requisitos [Saat80] [Kari89].
- *QFD* (Quality Function Deployment). Basada en la asignación numérica de prioridades respecto a una escala absoluta [Zult92].
- *Método de 100 Puntos.* Donde a cada participante se le dan 100 puntos para aplicar votando a favor de los requisitos más importantes para él [Leff03].
- *MoSCoW.* Agrupa los requisitos cualitativamente en cuatro grupos de importancia, que se negocian entre los involucrados/as, teniendo igual prioridad todos los requisitos pertenecientes a un grupo [Bark94].
- *Top-Ten.* Cada involucrado elige los diez requisitos que

considera de mayor prioridad y se arma una lista con los requisitos más importantes (sin ordenar), balanceando en ella la cantidad de requisitos elegidos por cada involucrado. Se asume que todos los involucrados/as tienen el mismo peso [Smit22].

- *Método de Wiegers.* Se basa en calcular una prioridad para cada requisito en base a cuatro atributos: beneficio para el cliente, penalidad si el requisito no se incluye, costo de implementación y riesgo técnico, asignando a cada atributo un valor entre 1 y 9, antes de calcular las prioridades se determina que todos los requisitos tengan el mismo nivel de abstracción y se considera si hay requisitos vinculados para incluir solo el requisito dominante [Wieg13].
- *Teoría W* (también denominada Win-Win). Donde cada involucrado da un ranking privadamente a los requisitos considerando aquellos que está dispuesto a abandonar y luego se realiza la negociación [Boeh89] [Park99].
- *Requirements Triage.* Incluye dar prioridades relativas a los requisitos, estimar los recursos necesarios para satisfacerlos y seleccionar un subconjunto de requisitos para optimizar la probabilidad de éxito [Davi03].
- *Juego de Planeamiento.* Usado en Programación Extrema sobre las historias de usuarios donde se agrupan los requisitos en grupos y luego se dan rankings dentro de cada grupo [Beck04].
- *SERUM.* Usa estimaciones de costo, beneficio, riesgo de desarrollo y reducción de riesgo operacional para dar prioridades [Gree99].

Priorizar los requisitos ayuda a una gestión eficaz de los recursos limitados como el tiempo, presupuesto, personal; permite maximizar el

valor para el cliente; mejora la toma de decisiones; es fundamental para reducir los riesgos durante la construcción del software y permite una mejor alineación del producto software con los objetivos organizacionales.

13.3 Trazabilidad de requisitos

Las palabras *rastreabilidad* (trackability) y *trazabilidad* (traceability) a menudo se usan indistintamente, pero no son sinónimos.

Trazabilidad → Habilidad de *dejar trazos*, donde un trazo corresponde a las relaciones directas entre artefactos o elementos del sistema (por ejemplo, un requisito que está vinculado a un caso de prueba). La forma más sencilla de registrar los rastros es construir una matriz de relaciones, como se puede observar en la Tabla 14. Siempre son valores booleanos.

Requisito	Caso Prueba1	Caso Prueba2	Caso Prueba3	Caso Prueba4	Caso Prueba5
REQ-001	1	0	0	0	0
REQ-002	0	1	1	0	0
REQ-003	0	0	0	1	0
REQ-004	1	0	0	0	0
REQ-005	0	0	0	0	1

Tabla 14 - Matriz de trazabilidad

> **IEEE610-1990,** *"Trazabilidad es el grado en que se puede establecer una relación entre dos o más productos del proceso de desarrollo, especialmente productos que tienen una relación predecesor-sucesor o maestro-subordinado entre sí; por ejemplo, el grado en que los requisitos y el diseño coinciden en un componente de software".*

> **Gotel and Finkelstein** (1994), *"Es la capacidad de describir y de seguir la vida de un requisito, tanto en dirección hacia adelante y hacia atrás, es decir, desde sus orígenes, a través de su desarrollo y especificación, a su despliegue y uso subsecuentes, y a través de períodos de refinamiento y de la iteración en curso en cualesquiera de estas fases".*
>
> **Moustafaev** (2014), *"La trazabilidad es la capacidad de vincular de manera explícita los requisitos del sistema con su implementación, asegurando que todos los elementos del proyecto estén alineados con las necesidades originales y proporcionando un historial rastreable de decisiones".*
>
> **Almeida & Carvalho** (2019), "*La trazabilidad es la capacidad de vincular los requisitos a lo largo de todo el ciclo de vida del proyecto, desde su origen hasta los entregables finales, lo que garantiza que los objetivos definidos inicialmente se cumplan".*

Varios autores han propuestos diferentes formas de trabajar [Palm96] [Jark98] [Wink10] [Mous14] [Ali18] [Wink18] [Alme19] [Beck19] [Ghos20] [Wohl20] [Silv21], entre otros, asegurando que la importancia de la trazabilidad es asegurar que se estén construyendo los artefactos correctos en cada fase del ciclo de vida de desarrollo del software y que los cambios en los requisitos sean correctamente incorporados.

La trazabilidad no solo se utiliza para administrar los cambios en los requisitos, sino que también es de fundamental ayuda para la verificación y validación y para el control del proceso de desarrollo [Palm96] [Davi99], pues facilita detectar conflictos utilizando los vínculos establecidos entre los elementos, posibilita asegurar que las

decisiones que se van tomando en la construcción de software sean consistentes con decisiones tempranas, y permite verificar que todos los requisitos han sido implementados en el software, entre otros usos.

Gotel and Finkelstein [Gote94] (ver Figura 39) introducen la distinción entre la trazabilidad previa a la especificación de requisitos (pre-RS) y la trazabilidad posterior a la especificación de los requisitos (post-RS):

- *Pre-Trazabilidad* (Pre-RS traceability). Se refiere a aquellos aspectos de la vida de un requisito antes de su inclusión en la ERS.
- *Post-Trazabilidad* (Post-RS traceability). Se refiere a aquellos aspectos de la vida de un requisito que resultan de la inclusión en la ERS.

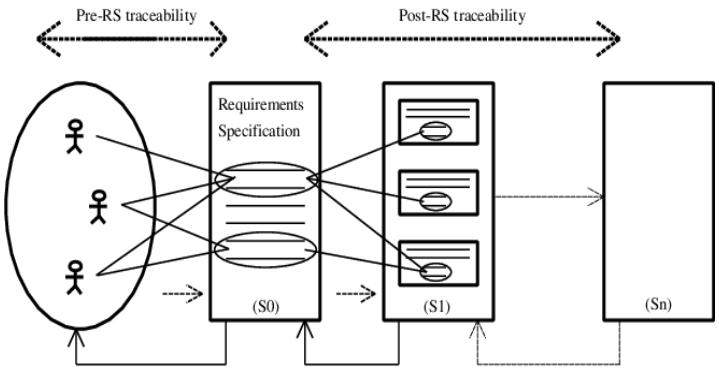


Figura 39 – Trazabilidad [Gotel94]

Un estudio posterior [Uzai08] en el cual se analizó qué definiciones se habían utilizado en un conjunto de trabajos publicados, determinó que el 80% habían citado a Gotel and Finkelstein.

Otros autores [Wier95] [Davi03] (ver Figura 40) clasifican la trazabilidad de la siguiente manera:

- *Trazabilidad hacia atrás* (Backward-from traceability). Vincula los requisitos con sus fuentes, que se encuentran en otros

documentos o de personas. Backward-from y Pre-RS traceability traceability pueden ocurrir al mismo tiempo. Por ejemplo, cuando se desea conocer la fuente de un cambio en un requisito, se rastrea desde el requisito hacia la FI que aceptó el cambio.

- *Trazabilidad hacia adelante* (Forward-from traceability). Vincula los requisitos con los componentes de diseño e implementación. Forward-from traceability y Post-RS traceability pueden ocurrir al mismo tiempo. Por ejemplo, cuando se detecta un error en un requisito y se rastrea hacia el código.
- *Trazabilidad desde atrás* (Backward-to traceability). Vincula los componentes de diseño e implementación con los requisitos.
- *Trazabilidad desde adelante* (Forward-to traceability). Vincula otros documentos a los requisitos relevantes. Estos otros documentos pueden ser, por ejemplo, manuales de funcionamiento que describen la funcionalidad del sistema.

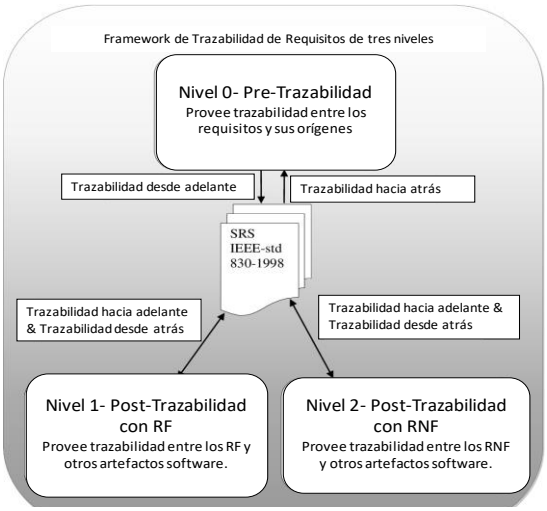


Figura 40 - Trazabilidad [Wang19]

Para Pinheiro [Pinh04], además de los cuatro tipos mencionados, incluye dos tipos adicionales:

- *Trazabilidad entre requisitos* (Inter-requirements traceability). Se refiere a las relaciones entre requisitos de un mismo documento.
- *Trazabilidad hacia afuera* (Extra-requirements traceability). Se refiere a las relaciones entre los requisitos y otros artefactos.

Estrictamente los inter trazabilidad no son rastros sino dependencias. Sin embargo, por cuestiones de costos se las suele tratar conjuntamente con los rastros porque se representan con las mismas estructuras. Se debe tener en cuenta que la trazabilidad de requisitos se puede tornar inevitable, ya que en sistemas medianos a complejos la cantidad de información que se debe manejar es muy grande y se hace necesario contar con un modelo de trazabilidad para gestionarla. En estos sistemas existe una red de relaciones bastante compleja. Es común que varios requisitos provengan de la misma fuente, así como que un solo requisito tenga más de una fuente. También es común tener un requisito derivado de varios otros, así como varios requisitos que se colapsan en uno solo.

La diversidad y la gran cantidad de información que existe cuando se desarrolla un sistema de software de gran tamaño, genera la necesidad de contar con un soporte automatizado para todo el desarrollo, incluida la trazabilidad. La trazabilidad asegura que cada requisito pueda ser rastreado desde la regulación específica hasta las pruebas y validaciones realizadas en el software, facilitando auditorías y asegurando el cumplimiento normativo.

Rastreabilidad → Habilidad de *seguir el rastro*, donde un rastro es el historial de acciones o la evidencia de cómo un artefacto ha sido manipulado o modificado a lo largo del tiempo (por ejemplo, un historial de versiones o cambios realizados en un requisito).

> **Pinheiro & Goguen** (1996), *"La rastreabilidad de los requisitos se refiere a la habilidad de definir, capturar y seguir las pistas dejadas por los requisitos sobre otros elementos del ambiente de desarrollo del software y las pistas dejadas por dichos elementos sobre los requisitos".*
>
> **Torres & Silva** (2017), *"La rastreabilidad es el proceso de documentar y seguir los enlaces entre los requisitos y los artefactos generados durante el ciclo de vida del proyecto, lo que permite identificar el impacto de los cambios y garantizar el cumplimiento con las especificaciones".*

La rastreabilidad [Pinh96] [Torre17] [Mish18] se utiliza para identificar rápidamente dónde se encuentra cada requisito en un momento específico, monitorear avances y detectar problemas de cumplimiento en tiempo real. Por ejemplo, en una fase de pruebas, si un RNF ("El sistema debe responder en menos de 2 segundos") no se cumple, la rastreabilidad permite rastrear rápidamente dónde se encuentra el problema. Los equipos de calidad pueden verificar el estado del requisito en las pruebas de rendimiento y asegurarse que los problemas se resuelvan antes de la entrega final del software.

Entonces, la *trazabilidad* asegura que se cumplan todos los requisitos a lo largo del desarrollo, mientras que la *rastreabilidad* permite identificar y monitorear el estado de cada requisito en cualquier momento del proceso.

13.4 Administración del cambio

Es un proceso para administrar los cambios de los requisitos e introducirlos en el proceso de construcción del software de una manera segura. Cuando se realiza un cambio que afecta a la Ingeniería de Requisitos, el ingeniero de requisitos realiza el análisis de impacto, evaluando cómo el cambio propuesto afectará los requisitos existentes, las dependencias y otras partes del sistema. Esto también incluye la evaluación de los riesgos asociados y el costo.
Paulk et al. [CMM93] incluye a la gestión de requisitos como una de las seis áreas de proceso clave para que una organización pueda pasar del nivel 1-inicial al nivel 2-repetible, estableciendo que:

- Se debe determinar el impacto del cambio antes de realizar el cambio.
- Los cambios se deben negociar y comunicar a los grupos que serán afectados por el cambio.
- Todo cambio debe ser rastreado hasta ser efectivamente cumplido.

Aunque no es el encargado directo de aprobar o gestionar el cambio, el ingeniero de requisitos actúa como facilitador y colaborador clave con otros roles, como gestor de proyecto o equipos de desarrollo, asegurando que todos los interesados comprendan las implicaciones del cambio. En este contexto la Ingeniería de Requisitos debe trabajar conjuntamente con la *gestión del cambio* [Kott12] [Pros20] [Lope20] y la *gestión de la configuración* [IEEE828] [Leon15]. La gestión del cambio proporciona un proceso formal para manejar modificaciones en

los requisitos y otros artefactos del proyecto y su objetivo es evaluar, aprobar, implementar y rastrear los cambios, minimizando el impacto en el alcance, tiempo y costo del proyecto. La gestión de la configuración garantiza que todos los artefactos del proyecto (incluidos los requisitos) estén bien identificados, versionados y sean consistentes. Su objetivo es mantener la integridad del sistema frente a cambios y asegurar que los elementos del sistema evolucionen de manera controlada.

Cómo interactúa la Ingeniería de Requisitos con la *gestión del cambio*:

- *Solicitud de cambio en los requisitos.* Cuando se solicita un cambio que afecta a los requisitos existentes o es un nuevo requisito, se debe introducir al proceso de Gestión del Cambio.
- *Análisis de impacto.* Durante este procedimiento el ingeniero de requisitos analiza qué requisitos específicos serán modificados y qué impacto tendrá en otros requisitos.
- *Aprobación o rechazo del cambio.* La Gestión del Cambio coordina con la Ingeniería de Requisitos para tomar decisiones basadas en el análisis de impacto. Si el cambio se aprueba, los requisitos afectados se actualizan y se genera una nueva versión.
- *Trazabilidad.* El ingeniero de requisitos actualiza los documentos y asegura que los cambios sean trazables hacia las implementaciones y pruebas asociadas.

La *gestión de configuración* es responsable de mantener la integridad y consistencia de los elementos del proyecto incluyendo los requisitos, para ello es necesario:

- *Establecer los requisitos como elementos de configuración.* Los requisitos son gestionados como parte de los elementos de configuración.
- *Integrar la trazabilidad.* La Gestión de Configuración integra la trazabilidad de los requisitos con el resto de los artefactos para garantizar que los cambios en los requisitos se reflejen adecuadamente en todo el sistema.
- *Controlar las versiones.* Cuando se actualiza un requisito debido a un cambio, la Gestión de Configuración registra la nueva versión y asegura que las versiones previas sean accesibles para referencia o auditoría.
- *Realizar auditorías.* La Gestión de Configuración verifica, mediante auditorías, que los requisitos actualizados se alineen con el sistema construido y que no existan inconsistencias entre los artefactos del proyecto.

Si bien la gestión del cambio y la gestión de configuración no son partes de la Ingeniería de Requisitos son complementarias y necesarias para administrar los cambios en los requisitos de manera ordenada y segura.

13.5 Asignación de requisitos

Una actividad de la Ingeniería de Requisitos y particularmente, de la gestión de requisitos, es la *asignación de requisitos* (requirements allocation). Su objetivo principal es distribuir, descomponer y asignar los requisitos del sistema a los elementos específicos de diseño o componentes del software, asegurando que cada requisito se pueda implementar de manera efectiva y trazable en la solución final. Este proceso conecta directamente los requisitos funcionales y no

funcionales con los elementos que los cumplirán en el diseño arquitectónico y, posteriormente, en el desarrollo. La asignación de requisitos se realiza generalmente con la participación activa del equipo de arquitectura del diseño. El proceso de Asignación de Requisitos es el siguiente:

- *Descomposición de requisitos.* Los requisitos del sistema se desglosan en requisitos más específicos y detallados que sean manejables a nivel de componentes. Este desglose permite identificar qué partes del sistema son responsables de cumplir cada uno de los requisitos de alto nivel.
- *Asignación a componentes o subsistemas*. Una vez descompuestos, los requisitos se asignan a los componentes adecuados. Por ejemplo, en el desarrollo de un sistema de software, un requisito de seguridad puede asignarse al módulo de autenticación, mientras que un requisito de velocidad de procesamiento puede asignarse al componente de procesamiento de datos.
- *Verificación y validación.* Una vez que los requisitos han sido asignados, es fundamental verificar y validar que cada componente cumple con su parte de los requisitos, y que, en conjunto, el sistema completo cumple con los requisitos de alto nivel.
- *Trazabilidad*. Se debe mantener una trazabilidad que conecte los requisitos de alto nivel con los requisitos asignados a los componentes, de manera que cualquier cambio en un requisito de alto nivel pueda rastrearse y gestionarse a través de todos los componentes que lo implementan.

- En la Figura 41 se muestra el proceso de asignación y su relación con el diseño. Se puede observar que en el Análisis de Requisitos se *analizan* los requisitos en términos funcionales y no funcionales y se descomponen en requisitos más específicos. Luego, son *asignados* a módulos, componentes, y relaciones en la arquitectura del software. Una vez asignados, se *verifica* si la arquitectura satisface los requisitos. Si no lo hace, se retroalimenta al Análisis de Requisitos para realizar los ajustes necesarios a los requisitos, generando un flujo bidireccional que muestra que los requisitos pueden influir en el diseño arquitectónico y, al mismo tiempo, que el diseño puede detectar problemas o limitaciones en los requisitos iniciales. Finalmente, la verificación permite asegurar que los requisitos asignados han sido correctamente incorporados en la arquitectura, o sea, que cubre todos los requisitos especificados y que no hay ambigüedades ni omisiones.

Cabe destacar que se debe mantener actualizado el modelo de trazabilidad para garantizar que cada requisito esté relacionado con un componente o módulo específico del sistema.

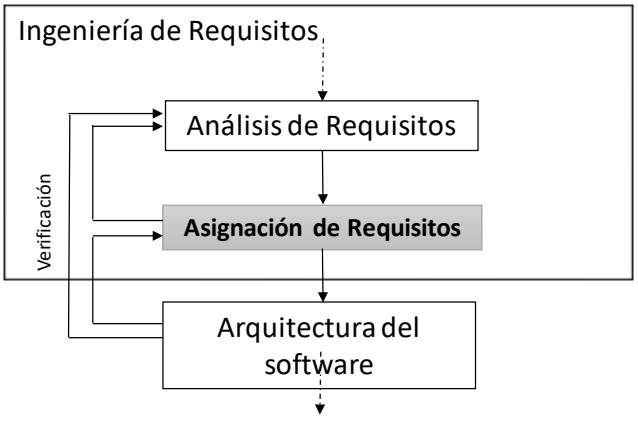


Figura 41 – Asignación de requisitos

La asignación de requisitos es un paso fundamental para pasar de una fase abstracta, centrada en *qué* debe hacer el sistema (IR), a una fase más concreta que detalla *cómo* se logrará esa funcionalidad en términos de arquitectura y componentes específicos (diseño). Esta asignación ayuda a definir la arquitectura de alto nivel del sistema y a estructurar el diseño en base a las necesidades del sistema, facilitando el trabajo de los equipos de diseño y desarrollo en los módulos o componentes específicos.

Referencias

[Ali18] Ali, S., Yue, T., & Briand, L. (2018). "Requirements traceability and its impact on system quality". Empirical Software Engineering, 23(4), 2343–2374. https://doi.org/10.1007/s10664-017-9574-0

[Alme19] Almeida, F. A., & Carvalho, J. A. (2019). "Requirements traceability in software projects: A systematic mapping study". Journal of Systems and Software, 157, 110393. https://doi.org/10.1016/j.jss.2019.110393

[Assa10] Assawamekin, N., Sunetnanta, T., & Pluempitiwiriyawej, C. (2010). "Ontology-based multiperspective requirements traceability framework". Knowledge and Information Systems, 25(3), 493-522. https://doi.org/10.1007/s10115-009-0259-2

[Avde16] Avdeenko, T. V., & Pustovalova, N. V. (2016). "The ontology-based approach to support the requirements engineering process". In 2016 13th International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE) (pp. 513-518). IEEE. https://doi.org/10.1109/APEIE.2016.7806406

[Bark94] Clegg, D. y Barker, R. (1994). "Case Method Fast-Track: A RAD Approach", Addison-Wesley.

[Beck04] K. Beck, and C. Andres. (2004). "Extreme Programming Explained: Embrace Change", Boston, MA: Addison-Wesley, 2º edición.

[Beck19] Becker, J., Kugeler, M., & Rosemann, M. (2019). "Process Management: A Guide for the Design of Business Processes". Springer.

[Boeh89] Boehm B. and Ross R. (1989). "Theory-W Software Project Management: Principles and Examples", IEEE TSE, Vol.15, Nº4, pp. 902-916.

[Brac90] Brackett J.W. (1990). "Software Requirements", (SEI-CM-19-1.2, ADA235642), Pittsburgh, Software Engineering Institute, Carnegie Mellon University.

[Cast14] Castañeda, V., Ballejos, L. C., Caliusco, M. L., & Galli, M. R. (2014). "The Use of Ontologies in Requirements Engineering". Global Journal of Researches in Engineering GJRE Classification, 10(2). Retrieved from https://www.researchgate.net/publication/228909488

[CMM93] Paulk, M. C., Curtis, B., Chrissis, M. B., & Weber, C. V. (1993). "Capability Maturity Model for Software", Version 1.1. Carnegie Mellon University, Software Engineering Institute. https://resources.sei.cmu.edu

[CMMI06] Software Engineering Institute. (2006). "Capability Maturity Model Integration", CMMI-DEV v1.2, CMU/SEI-2006-TR-008, Carnegie Mellon University, http://www.sei.cmu.edu/cmmi/, accedida el 14-10-2007.

[Davi03] Davis A. (2003). "The Art of Requirements Triage", IEEE Computer, Vol.36, Nº3, pp. 42-49.

[Davi99] Davis, A., Leffingwell, D. (1999) "Making Requirements Management Work For You", Crosstalk, The Journal of Defense Software Engineering, Vol.12, Nº4.

[Farf11] Farfeleder, S., Moser, T., Krall, A., Stålhane, T., Omoronyia, I., & Zojer, H. (2011). "Ontology-Driven Guidance for Requirements Elicitation" (pp. 212-226). https://doi.org/10.1007/978-3-642-21064-8_15

[Ghos20] Ghosh, S., Ali, N., & Arefeen, A. (2020). "Enhancing traceability in traditional software projects through systematic documentation practices". Journal of Software Engineering and Applications, 13(4), 195-210.

[Gote94] Gotel, O.C.Z., Finkelstein, A.C.W. (1994). "An analysis of the requirements traceability problem", ICRE'94, First IEEE International Conference on Requirements Engineering, IEEE

Computer Society Press, Colorado Springs, pp.94-101.

[Gree99] Greer D., Bustard D., and Sunazuka T. (1999). "Prioritisation of system changes using cost-benefit and risk assessments", Fourth IEEE International Symposium

[IEEE828] IEEE Standard for Configuration Management in Systems and Software Engineering (IEEE 828-2012). Institute of Electrical and Electronics Engineers.

[ISO29148] "Systems and software engineering — Life cycle processes — Requirements engineering". (2018). ISO/IEC/IEEE 29148. https://normasiso.org/norma-iso-29148/#google_vignette

[Jark98] Jarke, M. (1998). "Requirements tracing", Communications of the ACM, Vol.41, Nº12, pp.32-36.

[Kari89] Karlsson J., Wohlin C., and Regnell B. (1989). "An evaluation of methods for prioritizing software requirements", Information and Software Technology, Vol.39, Nº14-15, pp. 939-947.

[Kott12] Kotter, J. P. (2012). "Leading change". Harvard Business Review Press.

[Leff03] Leffingwell D., and Widrig D. (2003). "Managing Software Requirements - A unified approach". Addison-Wesley Object Technology Series, 2º edición.

[Leon15] Leon, A. (2015). "Software configuration management handbook". Artech House.

[Lope20] Lopez, L., & Salado, A. (2020). "A framework for integrating requirements traceability into systems engineering processes". Systems Engineering, 23(1), 50–64. https://doi.org/10.1002/sys.21516

[Mish18] Mishra, D., & Sharma, A. (2018). "Managing requirements and traceability in traditional project management frameworks". International Journal of Information Systems and Project Management, 6(4), 35-49.

[Mous14] Moustafaev, J. (2014). "Requirements Engineering for Project Managers: A Practical Guide to Requirements Management". CRC Press.

[Palm96] Palmer, J.D. (1996). "Traceability", en Software Engineering, editores M. Dorfman y R.H. Thayer, IEEE Computer Society Press, pp.266- 276.

[Park99] Park J., Port D., and Boehm B. (1999). "Supporting Distributed Collaborative Prioritization for Win-Win Requirements Capture and Negotiation", International Third World Multi-conference on Systemics, Cybernetics and Informatics (SCI'99), Vol.2, Orlando, FL, pp. 578-584.

[Parr15] Parreira, P. A., & Penteado, R. A. D. (2015). "Domain ontologies in the context of Requirements Engineering". In 2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA) (pp. 1-8). IEEE. https://doi.org/10.1109/AICCSA.2015.7507206

[Pinh96] Pinheiro, F.A.C., Goguen, J.A. (1996). "An object-oriented tool for tracing requirements", IEEE Software, Special issue of papers from ICRE'96, Vol.13, Nº2, pp.52-64.

[Pinh04] Pinheiro, Francisco A. C., (2004). "Requirements traceability", En el libro de Prado Leite, J.C.S., Doorn, J.H. (eds) Perspectives on Software Requirements. Kluwer Academic Publishers, EEUU, ISBN: 1-4020-7625-8, Capítulo 5.

[Pros20] Prosci. (2020). "Change management: The people side of change". Prosci Learning Center Publications.

[Saat80] Saaty, T.L. (1980) "The Analytic Hierarchy Process". McGraw-Hill.

[Silv21] Silva, D. R., Santos, T. S., & Freitas, V. S. (2021). "Importance of requirements traceability in traditional project management approaches". Journal of Project Management, 32(3), 185–196. https://doi.org/10.1016/j.jpm.2021.185196

[Sitt17] Sitthithanasakul, S., & Choosri, N. (2017). "Application of software requirement engineering for ontology construction". https://doi.org/10.1109/ICDAMT.2017.7905010

[Smit22] Smith, J. (2022). "Metodologías ágiles y priorización en Gestión de Proyecto", Editorial Proyectos Modernos.

[Torre17] Torres, C., & Silva, D. (2017). "A framework for requirements traceability in traditional project management". International Journal of Project Management, 35(3), 251-265.

[Trok18] Trokanas, N., Koo, L., & Cecelja, F. (2018). "Towards a Methodology for Reusable Ontology Engineering: Application to the Process Engineering Domain". In Computer Aided Chemical Engineering (Vol. 43, pp. 471-476). Elsevier. https://doi.org/10.1016/B978-0-444-64235-6.50084-X

[Uzai08] Uzair Akbar Raja and Kashif Kamran. (2008). "Framework for Requirements Traceability - TLFRT supporting pre-RS & post-RS traceability", tesis de Maestria, Suiza.

[Wieg13] Wiegers, K., & Beatty, J. (2013). "Software requirements", (3rd ed.). Microsoft Press.

[Wier95] Wieringa, R.J. (1995). "An introduction to requirements traceability", Reporte Técnico IR-389, Faculty of Mathematics and Computer Science, University of Vrije, Amsterdam.

[Wink10] Winkler, S., von Pilgrim, J. (2010). "A survey of traceability in requirements engineering and model-driven development". Softw Syst Model 9, 529–565. https://doi.org/10.1007/s10270-009-0145-0.

[Wink18] Winkler, S., & Pilgrim, J. (2018). A survey of traceability in requirements engineering and model-driven development. Software and Systems Modeling, 17(2), 553–581. https://doi.org/10.1007/s10270-016-0549-8

[Wohl20] Wohlrab, R., Knauss, E., Steghöfer, J. et al. (2020). "Collaborative traceability management: a multiple case study from the perspectives of organization, process, and culture. Requirements", Eng 25, 21–45. https://doi.org/ 10.1007/s00766-018-0306-1.

[Zult92] Zultner R. (1992). "Quality Function Deployment (QFD) for Software: Structured Requirements Exploration", en Total Quality Management for Software, eds Schulmeyer & McManus, NY: Van Nostrand Reinhold, pp. 297-317.

Capítulo 14

Ingeniería de Requisitos en ambientes ágiles

14.1 Enfoque tradicional y ágil

Básicamente, existen dos enfoques de construcción de software: tradicional y ágil. Cabe aclarar que existen múltiples orientaciones dentro de cada categoría. En modo general, ambos enfoques tienen similitudes y también presentan importantes diferencias (particularidades).

Las *similitudes* entre ambos enfoques se pueden percibir en la búsqueda en satisfacer las necesidades del cliente y usuarios; en el desarrollo de un sistema de software que presente la mayor calidad al menor costo posible; en la necesidad de participar a los involucrados para obtener la información del contexto de una manera segura, rápida y precisa; en identificar diferentes tipos de requisitos, como ser los funcionales y los no funcionales; en requerir algún tipo de registro de los requisitos para asegurar que sean claros y rastreables; en la necesidad de verificar y validar los requisitos para garantizar que sean correctos, completos y viables; en la necesidad de gestionar los cambios de los requisitos lo más rápido y seguro posible; entre otras. Se debe tener en cuenta que cada enfoque realiza lo antes mencionado en diferentes momentos del proceso y con una intensidad propia.

Las *particularidades* de cada enfoque que se describen a continuación, son generalizaciones y pueden tener diferencias según el modelo de proceso o la metodología utilizada. Los ***enfoques tradicionales*** de construcción de software, como se ha descripto en los capítulos anteriores, son convenientes en proyectos donde existe la presunción de que los requisitos pueden definirse sin mayores contratiempos. Aplica a contextos donde los cambios son moderados, el proyecto es grande y complejo y cuando se necesita un alto grado de control y documentación. En estos entornos de trabajo se valora la predictibilidad y la estructura sólida. Los requisitos pueden ser definidos al inicio del proyecto, como sucede en el Modelo en Cascada, o de manera evolutiva, como sucede en el Modelo Espiral o Incremental. Los requisitos se registran de manera clara y detallada en un documento de especificación (ERS) que define el alcance del proyecto, donde los requisitos pueden ser clasificados y priorizados. Esta documentación facilita la trazabilidad y mejora significativamente la gestión y el control del proyecto. La conformación de equipos de trabajos suele ser jerárquica, se enfocan en la ejecución de tareas específicas para cada fase del proyecto, con roles especializados y responsabilidad centralizada. La comunicación en el enfoque tradicional se establece entre el equipo de desarrollo y todos los involucrados/as. Algunos de los modelos de proceso del enfoque tradicional son, además de los ya mencionados, el Desarrollo basado en Componentes, Prototipado, Formal, Operacional, DDM, RAD, entre otras. Por otro lado, para que una metodología se encuadre como un ***enfoque ágil*** debe, en primera instancia, alinearse con los principios y valores establecidos en el Manifiesto Ágil [Beck01]. Este tipo de enfoque es ideal cuando existe una exigencia del negocio en poner el software rápidamente en

funcionamiento. El enfoque ágil [Cock02] [High02] [Sill05] [Ambl12] [Erle21] se centra en el desarrollo iterativo, incremental y colaborativo, con ciclos de trabajo cortos y retroalimentación frecuente. La construcción de software ágil se destaca en contextos de cambio constante, de requisitos inciertos, de proyectos que requieren la participación continua del cliente y entregas rápidas. Los requisitos se tratan como elementos evolutivos que pueden cambiar y adaptarse a medida que avanza el proyecto. No se construye un documento de especificación, sino que se gestionan, habitualmente, mediante historias de usuario que se administran y priorizan en cada sprint o iteración. Los equipos de trabajo suelen ser autónomos y colaborativos con roles multifuncionales y responsabilidades compartida. La comunicación se establece entre el equipo de desarrollo y un representante del cliente o un Product Owner que actúa como un puente con el resto de los involucrados/as. Esta característica es una ventaja para algunos autores ya que permite centralizar la información en una sola persona y hace el proceso menos confuso, mientras que para otros autores se convierte en una desventaja ya que se genera una dependencia con esa persona además de poder sesgar la información. En este enfoque la *calidad* del producto software puede verse afectada cuando es necesario priorizar la rapidez y la adaptación al cambio a ciertos elementos de calidad. Esto puede suceder en mercados altamente competitivos, cuando las condiciones externas cambian de forma repentina, cuando es más importante validar una idea o tecnología en el mercado que el mismo producto, etc. En este enfoque se encuentran Scrum, Kanban, Extreme Programming (XP), Lean Software Development, entre otras.

Con respecto a la gestión de los proyectos ambos enfoques utilizan el conocido *triángulo de oro de los proyectos* que involucra el alcance, el

tiempo y el costo visto en la sección 2.3. Las diferencias pueden observarse en la Figura 42.

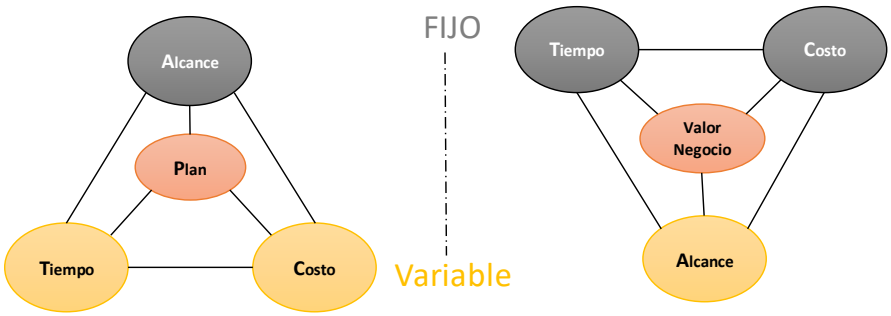


Figura 42 – Gestión de proyecto tradicional y ágil

En la parte izquierda del gráfico que representa el enfoque tradicional, el *alcance* se establece como fijo, es decir, no cambia a lo largo del proyecto. Esto implica que el equipo debe planificar en función de un objetivo definido. Las variables ajustables son el *tiempo* y el *costo*, lo que significa que, para cumplir con el alcance fijo, el proyecto podría necesitar más recursos o extender el tiempo de entrega. El *plan* se sitúa como el eje central que conecta las tres variables. En la parte derecha del gráfico que representa el enfoque ágil, el *tiempo* y el *costo* son las variables fijas, mientras que el *alcance* es flexible. Este enfoque permite ajustar el alcance para entregar el *valor de negocio* prioritario dentro de los límites establecidos de tiempo y presupuesto. El *valor de negocio* es el núcleo de este esquema, lo que sugiere un enfoque donde la prioridad radica en entregar valor continuamente en lugar de adherirse a un alcance rígido.

Indudablemente, ambos enfoques tienen fortalezas y debilidades. La efectividad del enfoque elegido dependerá, fundamentalmente, del grado de claridad y estabilidad de los requisitos; la naturaleza del proyecto; el nivel de complejidad e incertidumbre; la participación del

cliente; las restricciones de tiempo y presupuesto y la cultura y habilidades del equipo.

14.2 Tratamiento de los requisitos en metodologías ágiles

La perspectiva con la que se definen los requisitos varía en cada metodología ágil. A continuación, se describen brevemente algunas de ellas:

- *Scrum*. Es un marco ágil que organiza el trabajo en sprints (ciclos cortos de tiempo de 1 a 4 semanas) para entregar incrementos del producto funcionales. Se centra en equipos pequeños y multifuncionales. Promueve reuniones para asegurar el avance del proyecto [Schw97] [Pich10] [Suth14] [Schw20] [Derb20].

  Tratamiento de los Requisitos: por ser la metodología ágil con mayor penetración en la industria (ver estadísticas al final de esta sección), este punto se desarrolla en la sección 14.3.

  ✓ **Puntos clave:** Refinamiento incremental, backlog priorizado, enfoque en valor.

- *Extreme Programming* (XP). Enfatiza prácticas como la programación en pareja, las entregas frecuentes, las pruebas automatizadas y el diseño simple. Fomenta una comunicación cercana y continua entre los desarrolladores/as y el cliente [Beck99] [Berr02][Pinh03].

  Tratamiento de los Requisitos: Se usan Historias de Usuario escritas en tarjetas físicas o herramientas digitales. Las historias se desglosan en Tareas Técnicas. Se fomenta la comunicación continua con el cliente para cambios frecuentes. Se utilizan

pruebas automatizadas (TDD) para validar requisitos desde el inicio. Se priorizan requisitos en función de valor y riesgo.

✓ **Puntos clave:** Historias de usuario, retroalimentación constante, pruebas automatizadas.

- *Kanban*. Se basa en la visualización del trabajo en progreso a través de un tablero Kanban. Es más flexible que Scrum, ya que no requiere iteraciones fijas ni reuniones formales, lo que lo hace ideal para equipos con tareas de flujo continuo. Se basa en limitar el trabajo en progreso (WIP) para evitar la sobrecarga de tareas [Lada09] [Ande10].

  Tratamiento de los Requisitos: No hay un backlog tradicional, sino un flujo continuo de trabajo. Los requisitos ingresan a la columna To Do y avanzan a medida que hay capacidad. Se establecen límites en el trabajo en progreso (WIP) para evitar sobrecarga. Se basa en priorización constante y no en iteraciones fijas.

  ✓ **Puntos clave:** Flujo continuo, sin iteraciones fijas, requisitos adaptables.

- *Lean Software Development*. Está basado en los principios del Lean Manufacturing. Se centra en eliminar desperdicios y maximizar el valor entregado al cliente. Enfatiza la entrega rápida, el aprendizaje continuo y la mejora constante**.** Se enfoca en principios como la entrega Just-in-Time y la reducción del trabajo no necesario [Popp03] [Jane14].

  Tratamiento de los Requisitos: Se centra en minimizar desperdicios y entregar solo lo necesario. Los requisitos evolucionan a medida que se descubre el valor real del producto. Se fomenta la toma de decisiones lo más tarde posible

para mantener flexibilidad. Se optimiza el flujo de trabajo con iteraciones ligeras y mejora continua.

✓ **Puntos clave:** Eliminación de desperdicios, decisiones tardías, flexibilidad.

- *Feature-Driven Development* (FDD). Se centra en diseñar y construir funcionalidades específicas del sistema de manera iterativa. Utiliza cinco pasos clave: desarrollar un modelo general, construir una lista de características, planificar por característica, diseñar por característica y construir por característica [Coad00].

  Tratamiento de los Requisitos: Se define un modelo de dominio antes de desarrollar funciones. Los requisitos se organizan en funcionalidades ("features") en lugar de historias de usuario. Cada funcionalidad se desarrolla en un ciclo de cinco pasos: desarrollo de un modelo general; construcción de una lista de funcionalidades; planificación por funcionalidad; diseño por funcionalidad y construcción por funcionalidad.

  ✓ **Puntos clave:** Modelo de dominio, funcionalidades pequeñas, planificación estructurada.

- *Crystal*. Es una familia de metodologías ágiles (Crystal Clear, Crystal Orange, Crystal Red, etc.) que se adapta según el tamaño del equipo, la criticidad del proyecto y otros factores contextuales. Fomenta una comunicación cercana, un enfoque en personas y la entrega frecuente [Cock04].

  Tratamiento de los Requisitos: Se adapta según el tamaño del equipo y la criticidad del proyecto. Se enfoca en la comunicación directa y la colaboración del equipo. No define un formato rígido para requisitos, sino que los adapta según

contexto. Se prioriza la entrega de software funcional sobre la documentación extensiva.

✓ **Puntos clave:** Flexibilidad, comunicación abierta, adaptación al contexto.

- *Scaled Agile Framework* (SAFe). Es un marco para escalar metodologías ágiles en organizaciones grandes con múltiples equipos. Integra prácticas de Scrum, Kanban y Lean para coordinar equipos a nivel empresarial. Introduce roles adicionales como el Release Train Engineer y planes a nivel de programa [Cook21] [Scal21].

  Tratamiento de los Requisitos: Se organiza en niveles (Equipo, Programa, Portafolio). Los requisitos a alto nivel se expresan como Épicas, desglosadas en Features y luego en User Stories. Se utiliza un Program Backlog para gestionar los requisitos en equipos grandes. Se priorizan requisitos con el modelo WSJF (Weighted Shortest Job First). Se definen entregas coordinadas en Program Increments (PI) de varias iteraciones.

  ✓ **Puntos clave:** Jerarquía de requisitos, enfoque en escalabilidad, coordinación entre equipos.

La *Encuesta State of Agile* de 2023 ha informado que Scrum es utilizada en un 58% de las organizaciones, mientras que Kanban en un 7%, SAFE en un 6%, Lean en un 3% y un 2% se reparte entre otras metodologías como Crystal, FDD, etc. El 21% restante agrupa a las combinaciones [Bell21] como Scrumban, Scrum/XP, etc. y a las organizaciones que han optado por crear su propio marco de trabajo o no siguen ningún marco obligatorio. Este último grupo ha teniendo un crecimiento importante.

Las metodologías ágiles han ganado una gran aceptación en la industria del software debido a su flexibilidad, adaptación al cambio y enfoque iterativo. Sin embargo, el tratamiento de los requisitos es altamente cuestionable. Uno de los principales problemas de las metodologías ágiles es su enfoque informal hacia la gestión de requisitos. Sommerville [Somm20] hace referencia que las metodologías agiles generan una documentación mínima de los requisitos, confiando en la comunicación verbal y la interacción constante con los clientes. Si bien esto puede ser efectivo, en proyectos grandes y complejos donde participan múltiples partes interesadas, la falta de documentación estructurada puede generar ambigüedad, pérdida de conocimiento, dificultad en la trazabilidad de los cambios, entre otras. En metodologías ágiles, los requisitos evolucionan iterativamente a lo largo del desarrollo del software. Sin embargo, este enfoque puede llevar a cambios constantes que afectan la estabilidad del sistema, generando sobrecostos y retrasos imprevistos. La falta de una visión a largo plazo sobre los requisitos puede hacer que los equipos se enfoquen en soluciones inmediatas sin considerar la arquitectura general del sistema, lo que puede derivar en problemas de escalabilidad y mantenimiento a futuro [Boeh03]. En proyectos de gran escala, la gestión de requisitos se vuelve crítica debido a la interdependencia entre múltiples módulos, equipos y tecnologías. Las metodologías agiles, al priorizar entregas incrementales y feedback continuo, tiende a fragmentar la visión del producto, dificultando la gestión de la complejidad. También, existe un problema con la evaluación de calidad ya que se realiza de manera continua, lo que puede llevar a una falta de criterios sólidos para medir el éxito del producto software construido [Wieg13]. En síntesis, la gestión informal de los requisitos de las

metodologías agiles puede derivar en problemas de escalabilidad, costos imprevistos y dificultades para gestionar requisitos complejos.

14.3 Scrum y los requisitos

En el marco de trabajo de Scrum, como ya se mencionó, los requisitos se recopilan y mantienen en un Product Backlog que es una lista priorizada de historias de usuario o elementos que definen el producto y a medida que el equipo avanza, se realiza el refinamiento del Product Backlog que permite ajustar y mejorar los requisitos. Esto implica desglosar historias de usuario complejas, clarificar detalles y re-priorizar en función del valor y la retroalimentación obtenida. Luego, el equipo selecciona las historias de usuario de mayor prioridad para desarrollarlas en ese ciclo. Las revisiones y demostraciones de cada sprint permiten al equipo recibir opiniones tempranas sobre los requisitos implementados, lo que facilita realizar ajustes en las siguientes iteraciones.

En la definición de los requisitos se generan diferentes instancias: primero se describe una visión general del proyecto en *temas* (themes) que son requisitos de nivel muy general que pertenecen a la visión del proyecto. Estos se van refinando en una visión del producto, primero en épicas y luego, en Historias de Usuario [Cohn04]. Finalmente, las historias de usuario se asignan a los desarrolladores/as en cada sprint y se las denomina *tareas* (task). En la Tabla 15 se describe cada instancia.

Tipo	Descripción	Tamaño	Finalidad
Tema	Agrupación amplia de varias épicas o funcionalidades	Muy grande	Categorizar grandes áreas de trabajo

Tipo	Descripción	Tamaño	Finalidad
	relacionadas.		
Épica	Funcionalidad compleja que se descompone en varias historias de usuario.	Grande	Describir una iniciativa que requiere varias historias
Historia de Usuario	Funcionalidad específica desde la perspectiva del usuario final.	Pequeña	Describir una necesidad del usuario que se completa en un sprint
Tarea	Unidad de trabajo técnico concreta que forma parte de una historia de usuario.	Muy pequeña	Detallar una acción específica necesaria para completar una historia

Tabla 15 - Tipos de historias de usuario

En la Figura 43 se muestra un ejemplo de Tema, Épica, HU y Tarea utilizando un ejemplo del Sistema Gestión de ATM (ver Anexo).

Tema: Gestión de Números de OC
Descripción: El sistema debe proporcionar funcionalidades para gestionar los números de OC, incluyendo su generación, reserva y liberación, de modo que los procesos de compra sean eficientes y sin conflictos.
Épica: Reserva y gestión de Números de OC
Descripción: El sistema debe permitir a los responsables de compras reservar, gestionar y liberar números de OC de manera anticipada para garantizar la asignación ordenada y evitar conflictos durante el proceso de compra.
Historia de Usuario:
Título: Reservar números de OC **Como** responsable de compras (usuario), **quiero** reservar un número de OC, **para** asegurar que ciertos clientes especiales tengan un número de OC asignado para próximas compras sin conflictos.
Criterios de Aceptación:

1. El usuario debe poder acceder a la funcionalidad de reservar números de OC en el sistema solo si está autorizado en el sistema. 2. El sistema debe mostrar un rango de números de OC a reservar (desde el "último número utilizado +1" hasta 10 números posteriores). 3. El sistema debe mostrar el rango de números de OC y no permitir seleccionar aquellos números de OC previamente reservados. 4. El sistema debe permitir seleccionar los números de OC a reservar. 5. El sistema debe controlar que no se superen los 3 números de OC por mes. 6. El sistema debe registrar el responsable, fecha, hora y números de OC reservados. 7. …
***Tarea 1**:* Crear interfaz de reserva de números de OC **Descripción:** Desarrollar la interfaz de usuario donde el responsable de compras pueda seleccionar los números de OC a reservar. **Criterios de finalización:** La interfaz permite la selección y muestra el listado de números reservados.
Tarea 2: Implementar la generación automática de números de OC **Descripción:** Programar la funcionalidad que genere números de OC únicos y los reserve en el sistema. **Criterios de finalización:** Los números de OC se generan automáticamente según la cantidad seleccionada y se registran como "Reservados".
***Tarea 3**:* Registrar responsable, fecha, hora y números de OC de la reserva **Descripción:** Desarrollar la funcionalidad que registre automáticamente el responsable, fecha, hora y números de OC reservados. **Criterios de finalización:** Los datos se almacenan en la base de datos y es visible para el usuario.
Tarea 4: Crear un listado de números de OC reservados **Descripción:** Implementar una vista en la que el responsable de compras pueda ver los números de OC reservados. **Criterios de finalización:** El listado muestra todos los números reservados y cuales fueron liberados.
Tarea 5: Implementar la funcionalidad para liberar números de OC reservados **Descripción:** Desarrollar una opción para liberar los números de OC reservados que no han sido asignados en 20 días, devolviéndolos a la lista de números disponibles. **Criterios de finalización:** El sistema permite liberar números de OC, y los números liberados quedan disponibles nuevamente para ser reservados o asignados.

Figura 43 - Ejemplo de tema, épica, HU y tareas (Sistema Gestión de ATM)

Cuándo y cómo trata Scrum a los requisitos del software:

- Al inicio del proyecto, se crea el Product Backlog que es una lista priorizada de todas las funcionalidades, características y

tareas que se desean del producto. Los requisitos en el Product Backlog se organizan en historias de usuario.

- El Product Owner gestiona el Product Backlog, priorizando y ajustando las historias de usuario según el valor que aportan al cliente y al negocio.
- Al comienzo de cada sprint, se lleva a cabo la Sprint Planning. En esta reunión, el equipo de desarrollo y el Product Owner seleccionan las historias de usuario de mayor prioridad del Product Backlog para incluirlas en el Sprint Backlog, definiendo los requisitos específicos que se trabajarán en ese sprint. Los requisitos seleccionados se desglosan en tareas manejables para que el equipo pueda completarlas durante el sprint.

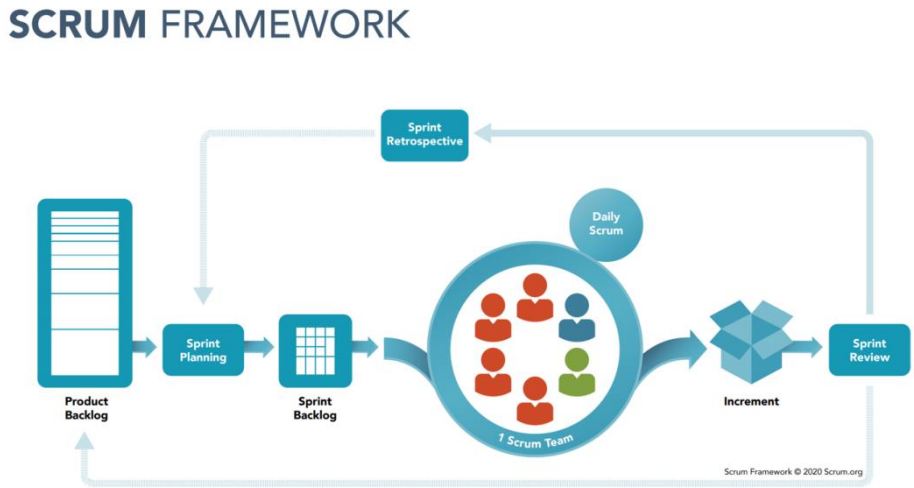


Figura 44 - Metodología Scrum[16]

- A lo largo del proyecto, el Product Backlog es revisado y refinado continuamente. Este proceso, conocido como backlog

[16] https://www.scrum.org/resources/blog/que-es-scrum

grooming o refinamiento, consiste en ajustar, detallar y repriorizar las historias de usuario.

- El Product Owner y el equipo pueden modificar los requisitos en función de la retroalimentación obtenida y los cambios en las necesidades del cliente o del mercado.
- Durante el sprint, se realizan las Daily Scrum para monitorear el avance y ajustar, de ser necesario, el enfoque. Las Daily permiten que el equipo se coordine para cumplir los requisitos seleccionados y solucione posibles bloqueos o ajustes.
- Al final de cada sprint, se realiza un Sprint Review donde el equipo presenta el trabajo completado al Product Owner y otras partes interesadas. Durante esta revisión, el Product Owner y los interesados proporcionan retroalimentación sobre los requisitos implementados, lo cual puede dar lugar a ajustes o a la creación de nuevos requisitos en el Product Backlog.

Evento	Propósito	Duración máxima
Sprint	Contenedor de todos los eventos y trabajo.	1-4 semanas.
Sprint Planning (Planificación del sprint)	Planificar qué y cómo se trabajará en el Sprint.	8 horas para un Sprint de un mes.
Daily Scrum (Diaria)	Coordinar y sincronizar al equipo diariamente.	15 minutos.
Sprint Review (Revisión del sprint)	Presentar el incremento y recibir retroalimentación.	4 horas para un Sprint de un mes.
Sprint Retrospective (Retrospectiva del sprint)	Reflexionar y mejorar el proceso del equipo.	3 horas para un Sprint de un mes.

Tabla 16 – Eventos de Scrum

Todos estos eventos están diseñados para proporcionar un flujo continuo de inspección y adaptación durante el proyecto. Mientras que

el *Sprint Planning* establece el marco de trabajo para el sprint, la *Daily Scrum* asegura que el equipo esté alineado y el trabajo progrese. Al final del sprint, el *Sprint Review* evalúa el resultado, mientras que el *Sprint Retrospective* mejora el proceso para los próximos ciclos.

Referencias

[Ambl12] Ambler, S. W., & Lines, M. (2012). "Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise", IBM Press.

[Ande10] Anderson, D. J. (2010). "Kanban: Successful evolutionary change for your technology business", Blue Hole Press.

[Beck99] Beck, K. (1999). "Extreme programming explained: Embrace change". Addison-Wesley.

[Beck01] Beck, K. et al. (2001). "Manifesto for agile software development". Agile Alliance. https://agilemanifesto.org

[Bell21] Belling, S. (2021). "Succeeding with Agile Hybrids: Project Delivery Using Hybrid Methodologies", Apress. https://doi.org/10.1007/978-1-4842-6461-4

[Berr02] Berry, D. (2002). "The Inevitable Pain of Software Development, Including of Extreme Programming, Caused by Requirements Volatility", International Workshop on Time-Constrained Requirements Engineering (TCRE'02), Essen, Alemania, http://www-di.inf.puc-rio.br/~julio/tcre-site/p2.pdf

[Boeh03] Boehm, B., & Turner, R. (2003). "Balancing Agility and Discipline: A Guide for the Perplexed". Addison-Wesley.

[Coad00] Coad, P., & Nicola, P. (2000). "Feature-driven development: A practical guide to agile software delivery". Prentice Hall.

[Cock02] Cockburn, A. (2002). "Agile Software Development", Addison-Wesley.

[Cock04] Cockburn, A. (2004). "Crystal clear: A human-powered methodology for small teams". Addison-Wesley.

[Cohn04] Cohn, M. (2004). "User stories applied: For agile software development", Addison-Wesley.

[Cook21] Cook, T. A. (2021). "Practical Agile: Scaling Agile in Organizations", McGraw-Hill Education.

[Derb20] Derby, E., & Larsen, D. (2020). "Agile Retrospectives: Making Good Teams Great", Pragmatic Bookshelf.

[Erle21] Erlenhov, L., de Oliveira Neto, F. G., & Leitner, P. (2021). "How to Fail a Software Project: A Review of the Literature on Agile Methodologies", Most Common Pitfalls. Journal of Systems and Software, 178, 110971.

[Jane14] Janes, A., Succi, G. (2014). "Lean Software Development in Action". In: Lean Software Development in Action. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-00503-9_11

[High02] Highsmith, J. (2002). "Agile Software Development Ecosystems", Addison-Wesley.

[Lada09] Corey Ladas. (2009). "Scrumban - Essays on Kanban Systems for Lean Software". ISBN-10: 0578002140 ISBN-13: 978-0578002149

[Pich10] Pichler, R. (2010). "Agile product management with Scrum: Creating products that customers love". Addison-Wesley Professional.

[Pinh03] Pinheiro, F. (2003). "Requirements honesty". *Requirements Enginnering* 8, 183–192 (2003). https://doi.org/10.1007/s00766-003-0165-1

[Popp03] Poppendieck, M., & Poppendieck, T. (2003). "Lean software development: An agile toolkit", Addison-Wesley.

[Scal21] Scaled Agile, Inc. (2021). "*SAFe 5.0 framework"*. https://scaledagileframework.com/

[Schw97] Schwaber, K. (1997). "SCRUM Development Process". In: Sutherland, J., Casanave, C., Miller, J., Patel, P., Hollowell, G. (eds) Business Object Design and Implementation. Springer, London. https://doi.org/10.1007/978-1-4471-0947-1_11

[Schw20] Schwaber, K., & Sutherland, J. (2020). "The Scrum Guide: The definitive guide to Scrum: The rules of the game". Scrum.org.

[Sill05] Sillitti, A., & Succi, G. (2005). "Requirements Engineering for Agile Methods". In book Engineering and Managing Software Requirements". DOI:10.1007/3-540-28244-0_14. pp. 309-326.

[Somm20] Sommerville, I. (2020). "Software Engineering". (10th ed.). Pearson.

[Suth14] Sutherland, J., & Sutherland, J. J. (2014). “Scrum: The art of doing twice the work in half the time”. Crown Business.

[Wieg13] Wiegers, K., & Beatty, J. (2013). “Software Requirements” (3rd ed.). Microsoft Press.

ÍNDICE DE FIGURAS

Índice de tablas

ACRÓNIMOS

ADOO / OOAD	Análisis y Diseño Orientado a Objetos / Oriented-Object Analysis and Design
AHP	Analytic Hierarchy Process
AS / SA	Análisis de Sistemas / Systems Analysis
ASD	Adaptive Software Development
AUP	Agile Unified Process
ATM	Automated Teller Machine (Cajero Automático)
BPMN	Business Process Model and Notation
CBD	Component Based Development
CD	Continue Distribution (distribución continua)
CI	Continue Integration (integración continua)
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
COTS	Commercial Off-the-Shelf
CU / UC	Caso de Uso / Use Case
D SDM	Método de desarrollo de sistemas dinámicos
DDM	Desarrollo Dirigido por Modelos
DEO	Discrepancias, Errores y Omisiones
DevOps	Development + Operation
DFD	Data Flow Diagram / Diagrama de Flujo de Datos
DII / IID	Desarrollo Iterativo Incremental / Iterative and Incremental Development
DoD	U.S. Departament of Defense
DOORS	Dynamic Object Oriented Requirements Systems
DSDM	Dynamic Systems Development Method

DSL	Domain-Specific Languages
ERS / SRS	Especificación de Requisitos de Software / Software Requirements Specification
FDD	Feature-Driven Development
FI	Fuente de Información
GQM	Goal Question Metrics
HU / US	Historia de Usuario / User Story
IA	Inteligencia Artificial
iA	Información Adelantada
ID	Iteration Development
iE	Información Extemporánea
IEEE	Institute of Electrical and Electronic Engineering
IFIP	International Federation for Information Processing
IoT	Internet of Things
IR / RE	Ingeniería de Requisitos / Requirements Engineering
IS / SI	Ingeniería de Software / Software Engineering
ISO	International Standards Organization
iT	Información Tardía
JAD	Join Application Development
LN / NL	Lenguaje Natural / Natural Language
LSD	Lean Software Development
NSQE	National Software Quality Experiment
PO	Product Owner
ONG	Organización No Gubernamental
OO	Orientación a Objetos
OpenUP	Open Unified Process
OTAN	Organización del Tratado del Atlántico Norte
PDSA	Plan-Do-Study-Act

PMI	Project Management Institute
PMI Agile	Project Management Institute Agile
QA	Quality Assurance
QFD	Quality Function Deployment
RAD	Rapid Application Development
RAE	Real Academia Española
RF	Requisito Funcional
RNF	Requisito No Funcional
RS	Requirements Specification
RUP	Rational Unified Process
SADT	Structured Analysis and Design Technique
SDLC	Software Develop Life Clicle
SQL	Structured Query Language
SysML	Systems Modeling Language
TI / IT	Tecnología de la Información / Information Technology
UdeD	Universo de Discurso
UML	Unified Modeling Language
UX	User Experience / Experiencia de usuario
V&V	Verificación y Validación
VDM	Vienna Development Method
XP	Extreme Programming

Anexo - ERS del Sistema Gestión de ATM[17] (ISO 29148:2018)

Nota: la ERS que sigue es una parte del documento original. Su objetivo es ejemplificar la relación del documento de especificación con modelos previos (ver Figura 23 y Tabla 6). Además, se muestra el uso de los atributos de los requisitos y la utilización de la norma.

1. Introducción

1.1 Propósito

Este documento tiene como objetivo describir los requisitos del *Sistema Gestión de ATM*. Este sistema responde a la llegada de los cajeros automáticos desde el exterior, la carga del software para su configuración, las pruebas de calidad y la instalación en el cliente. Los cajeros son solicitados por los clientes (supermercados, bancos, etc.), se genera un pedido (OC[18]) y se solicita la fabricación en St. Petersburg (EEUU) a demanda. Una vez que llegan a Argentina se les instala la Configuración según el modelo, esto permite que el ATM se ponga operativo. Luego, se le instala el software frontend, o sea, el que permite la interacción con la gente. Se prueba su correcto funcionamiento y se emplaza en el cliente. Todo este proceso requiere tanto de un circuito administrativo como del técnico. El sistema debe permitir realizar el seguimiento de ambos circuitos, o sea, desde el pedido del cliente (OC) hasta que es puesto en funcionamiento. Es fundamental que el sistema informe en qué fase se encuentra un pedido. Este seguimiento deberá ser accesible tanto por el cliente como el personal de la empresa. El cliente accederá vía una aplicación móvil y el personal interno por una intranet.

El presente documento de especificación está dirigido a los responsables del área comercial y técnica de la organización como así también, a aquellos que requieran una descripción detallada de las funcionalidades y restricciones que deberá contemplar el sistema informático.

El presente documento responde al estándar ISO 29148:2018 "*Systems and software engineering. Life cycle processes. Requirements engineering"*.

1.2 Alcance

[17] **ATM=** Automated Teller Machine (en español Cajero Automático)

[18] **OC=** Orden de Compra

El punto inicial del sistema es el pedido del cliente (CO) que se realizará por una app móvil, la cual se conectará al sistema central para realizar el seguimiento del ATM. El sistema registrará la carga del pedido, el seguimiento manual en fabrica, la configuración del ATM, la instalación del software y el traslado e instalación en el cliente. Finalmente, si es necesario, se registrará el mantenimiento post venta. En esta versión del software no se incluye la etapa de fabricación de los ATM en St. Petersburg, ni el seguimiento del traslado desde EEUU a Argentina, ni el mantenimiento.

Los beneficios que aportará el sistema a los procesos, serán:

- ✓ Optimización de los procedimientos administrativos.
- ✓ Confiabilidad y oportunidad de la información.
- ✓ Alto grado de disponibilidad de la información, para la realización de las tareas y para la toma de decisiones.
- ✓ Optimización de los recursos disponibles.
- ✓ Mejor atención a los clientes.

1.3 Definiciones, acrónimos y abreviaturas

En el ítem 4.1 se hace referencia al Léxico Extendido del Lenguaje para Requisitos del Software (LEL_R), el cual es un glosario que contiene definiciones de términos utilizados en modelos previos y en el presente documento. Contiene definiciones, siglas utilizadas, acrónimos, sinónimos y eventuales abreviaciones del vocabulario empleadas en el contexto que rodeará al sistema de software.

1.4 Referencias

Este documento ha sido confeccionado teniendo en cuenta la documentación de cada sector y el soporte digital que cada uno ha generado (Excel y bases de datos Access). Se han realizado entrevistas con responsables de cada sector que han sido documentadas y ratificadas mediante validaciones continuas. Todas las entrevistas han sido grabadas y luego, desgrabadas.

Los anexos adjuntos a este documento tienen toda la documentación de respaldo.

1.5 Descripción general del documento

Este documento está organizado de la siguiente manera: en la sección 2 se describen los requisitos funcionales; en la sección 3 se describen los requisitos no funcionales y en los anexos se presentan los modelos que dan soporte a este documento.

---

2. Descripción general del sistema

2.1 Perspectiva del producto

Para la carga y seguimiento de los ATM por parte del cliente, se realizará con una app móvil la cual registrará la información del

pedido. El sistema central conectará la administración, la fabricación y el área de soporte de la organización para registrar todos los pasos realizados en cada ATM en particular. Esto permitirá hacer las consultas de seguimiento y obtener estadísticas de errores, tiempos por áreas, satisfacción del cliente. El sistema operará solo con personal autorizado.

2.2 Funcionalidad del producto

Las funcionalidades que proveerá el sistema se enumeran a continuación:

- Carga del pedido y seguimiento del mismo por parte del cliente.
- Seguimiento de una OC desde que se crea hasta su instalación en el cliente. Se deberá registrar cada paso realizado relacionado con la OC (personal asignado, tiempo insumido en cada paso, problemas detectados, resolución de problemas, etc.).
- Mesa de ayuda para atención de reclamos del cliente por problemas en la instalación y mantenimiento post venta.

2.3 Características del usuario

El sistema debe contemplar diferentes accesos a funciones y datos.

El sistema estará disponible según los permisos otorgados a los siguientes grupos:

- Personal Administrativo (por jerarquía).
- Personal técnico (por área y jerarquía).
- Clientes (potenciales, activos y no activos).

Dentro de cada grupo de roles se asignarán sub roles para acceder a la información.

2.4 Restricciones

El sistema debe contemplar un mecanismo de alertas. Estas alertas deben dar seguridad que ante un problema o demora los responsables de cada área conozcan que se está en espera de resolución. Es de vital importancia asegurar que los tiempos asignados a cada paso se cumplan.

El sistema debe administrar permisos. Las responsabilidades permitidas en el sistema serán asignadas a través de un mecanismo de autorizaciones.

El sistema debe garantizar la seguridad y privacidad de los datos.

2.5 Suposiciones y dependencias

La información que procese el sistema se relacionará con la app móvil del Cliente, informando al mismo de los pasos realizados, los restantes y los problemas existentes.

3. Requisitos específicos

3.1 Requisitos funcionales

ID: REQ-001
Descripción: El sistema debe permitir reservar números de OC (ver Tabla 6).
Fundamento: Es esencial para las operaciones administrativas y comerciales del sistema.
Origen: Solicitud del área de Administración y Ventas para gestionar números de OC en ciertas operaciones comerciales.
Dependencia: Depende de la creación del sistema de autenticación de usuarios y gestión de permisos.
Dificultad: Alta.
Prioridad: Alta.

ID: REQ-002
Descripción: El sistema debe liberar los números de OC reservados cuando han pasado 15 días sin uso (ver Tabla 6).
Fundamento: Una vez pasados los 15 días ya no se pueden ingresar al circuito administrativo y es necesario cancelar esos números, pero esta cancelación está aceptada por la auditoria como una excepción y no como regla.
Origen: Solicitud del área de Administración para asegurar la consistencia de la información de compras.
Dependencia: Depende de que el sistema permita reserva de Números de OC habilitado.
Dificultad: Alta.
Prioridad: Media.

ID: REQ-003
Descripción: El sistema debe calcular un rango de números de OC donde el mínimo es el "último número de OC utilizado +1" y 10 números posteriores disponibles (Ej. ultimo número de OC = 14, el sistema muestra de 15 a 25 siempre que en ese rango no existan números previamente reservados) y mostrarlos en pantalla (ver Tabla 6).
Fundamento: Esta funcionalidad es clave para que se pueda realizar la selección de números válidos sin duplicaciones.
Origen: Solicitud del área de Administración que los números los gestione automáticamente el sistema de acuerdo con el flujo de trabajo de la empresa.
Dependencia: Proceso central para la reserva de números de OC, solicitado por el área operativa.

Dificultad: Media.
Prioridad: Media

ID: REQ-004
Descripción: El sistema debe permitir que el responsable de compras seleccione los números de OC que desea reservar (ver Tabla 6).
Fundamento: Es responsabilidad de compras reservar números de OC para darle servicio personalizado a ciertos clientes especiales.
Origen: Proceso central para la reserva de números de OC, solicitado por el área operativa.
Dependencia: Depende de la correcta asignación de los números de OC disponibles.
Dificultad: Alta.
Prioridad: Alta.

ID: REQ-005
Descripción: El sistema debe controlar que no se supere el número de reservas permitido por mes (ver Tabla 6).
Fundamento: Esta limitación obliga a restringir esta acción solo a casos particulares.
Origen: Proceso central para la reserva de números de OC, solicitado por el área legales.
Dependencia: Depende de la cantidad de reservas realizadas por mes.
Dificultad: Alta.
Prioridad: Media.

ID: REQ-006
Descripción: El sistema debe mostrar los números de OC seleccionados y solicitar confirmación de la reserva (ver Tabla 6).
Fundamento: Es necesario asegurar que la reserva que se está realizando es absolutamente necesaria porque limita las próximas reservas.
Origen: Mejora de la experiencia del usuario, solicitado por el equipo de UX para evitar errores en el proceso de reserva.
Dependencia: Depende de la funcionalidad de selección de números de OC y del control de límite de reserva.
Dificultad: Media.
Prioridad: Media.

ID: REQ-007

Descripción: El sistema debe registrar el responsable, fecha, hora y números de OC reservados (ver Tabla 6).
Fundamento: Es indispensable tener la trazabilidad de las reservas.
Origen: Proceso central para la reserva de números de OC, solicitado por el área operativa.
Dependencia: Depende de la funcionalidad de asignar un rango y números de OC disponibles.
Dificultad: Alta.
Prioridad: Alta.

ID: REQ-008
Descripción: Si sistema debe mostrar un mensaje de éxito cuando la operación de reserva finalice (ver Tabla 6).
Fundamento: Mejora la experiencia del usuario, solicitado por el equipo de UX, para mostrarle al usuario que la acción solicitada finalizo con éxito.
Origen: Solicitud del área de Administración para asegurar la consistencia de la información de compras.
Dependencia: Depende del correcto registro de las reservas.
Dificultad: Media.
Prioridad: Media.

ID: REQ-009
Descripción: El sistema debe permitirle solo al responsable de compras liberar números de OC reservados previamente (ver Tabla 6).
Fundamento: Los clientes especiales deben tener asegurada la compra mensual.
Origen: Solicitud del área de Administración para asegurar la consistencia de la información de compras.
Dependencia: Depende de la funcionalidad de reserva de números de OC.
Dificultad: Alta.
Prioridad: Media.

…

3.2 Requisitos no funcionales

3.3 Interfaces externas

ID: REQ-068
Descripción: El sistema debe tener acceso a libros/hojas/celdas de Excel y poder interactuar con la información que tienen.

Fundamento: El sistema debe poder aprovechar las funcionalidades de Excel para automatizar algunos procesos que los usuarios utilizan en dicha herramienta.
Origen: Documento técnico del cliente.
Dependencias: Integración con Microsoft Excel y compatibilidad con los objetos de Excel.
Dificultad: Media.
Prioridad: Alta.

ID: REQ- 069
Descripción: El sistema debe ser accesible tanto desde una PC como desde teléfonos móviles con SO Android versión 13 o superior.
Fundamento: Asegurar que los usuarios puedan acceder al sistema desde múltiples plataformas.
Origen: Necesidad del cliente para movilidad y accesibilidad.
Dependencias: Desarrollo de una interfaz adaptativa que funcione correctamente en ambas plataformas.
Prioridad: Media.

ID: REQ-070
Descripción: El sistema debe permitir la firma digital de documentos, conforme a la normativa de firma electrónica vigente.
Fundamento: Asegurar la autenticidad de los documentos y la verificación de los usuarios.
Origen: Requerimiento de cumplimiento regulatorio.
Dependencias: Compatibilidad con soluciones de firma digital aprobadas.
Dificultad: Alta.
Prioridad: Alta.

3.4 Requisitos de rendimiento

ID: REQ-084
Descripción: El sistema debe controlar que la operación de reserva de números de OC no supere los 90 segundos (ver Tabla 6).
Fundamento: Garantiza que el sistema funcione de manera eficiente y no bloquee otras operaciones críticas relacionadas con los números de OC.
Origen: Solicitud del área de operaciones y administración, debido a la necesidad de evitar retrasos y bloqueos durante el uso del sistema, especialmente cuando hay varios usuarios simultáneos.
Dependencias: Requiere una optimización adecuada en el acceso a la base de datos y la lógica de reserva de números.

Dificultad: Alta.
Prioridad: Alta.

ID: REQ-085
Descripción: El sistema debe generar alertas en diferentes puestos de trabajo. Si una alerta no es atendida en un plazo máximo de 2 horas, el sistema debe escalarla a un nivel jerárquico superior enviando información por correo electrónico y WhatsApp.
Fundamento: Asegurar la respuesta rápida a situaciones críticas y la notificación a niveles superiores cuando se superan los tiempos de atención.
Origen: Política interna.
Dependencias: Módulos de notificación y alertas.
Dificultad: Alta.
Prioridad: Alta.

3.5 Restricciones de diseño

ID: REQ-086
Descripción: El sistema debe generar un esquema de avisos para alertar de posibles problemas operacionales o fallas.
Fundamento: Mejorar la proactividad y evitar interrupciones inesperadas del servicio.
Origen: Directrices de diseño del cliente.
Dependencias: Sistema de monitoreo y alerta.
Dificultad: Media.
Prioridad: Alta.

ID: REQ-087
Descripción: El sistema debe integrarse con los sistemas existentes de fábrica en EEUU y del área administrativa en Argentina.
Fundamento: Minimizar costos y aprovechar las infraestructuras tecnológicas actuales.
Origen: Política de integración de sistemas.
Dependencias: API de los sistemas existentes.
Dificultad: Media.
Prioridad: Alta.

3.6 Atributos de calidad

ID: REQ-088
Descripción: El sistema debe registrar la información de acuerdo con los estándares utilizados en St. Petersburg (ISO 9001).
Fundamento: Asegurar que el sistema cumpla con los estándares internacionales de calidad.
Origen: Certificación ISO 9001.

Dependencias: Auditoría de calidad del sistema.
Dificultad: Alta.
Prioridad: Alta.

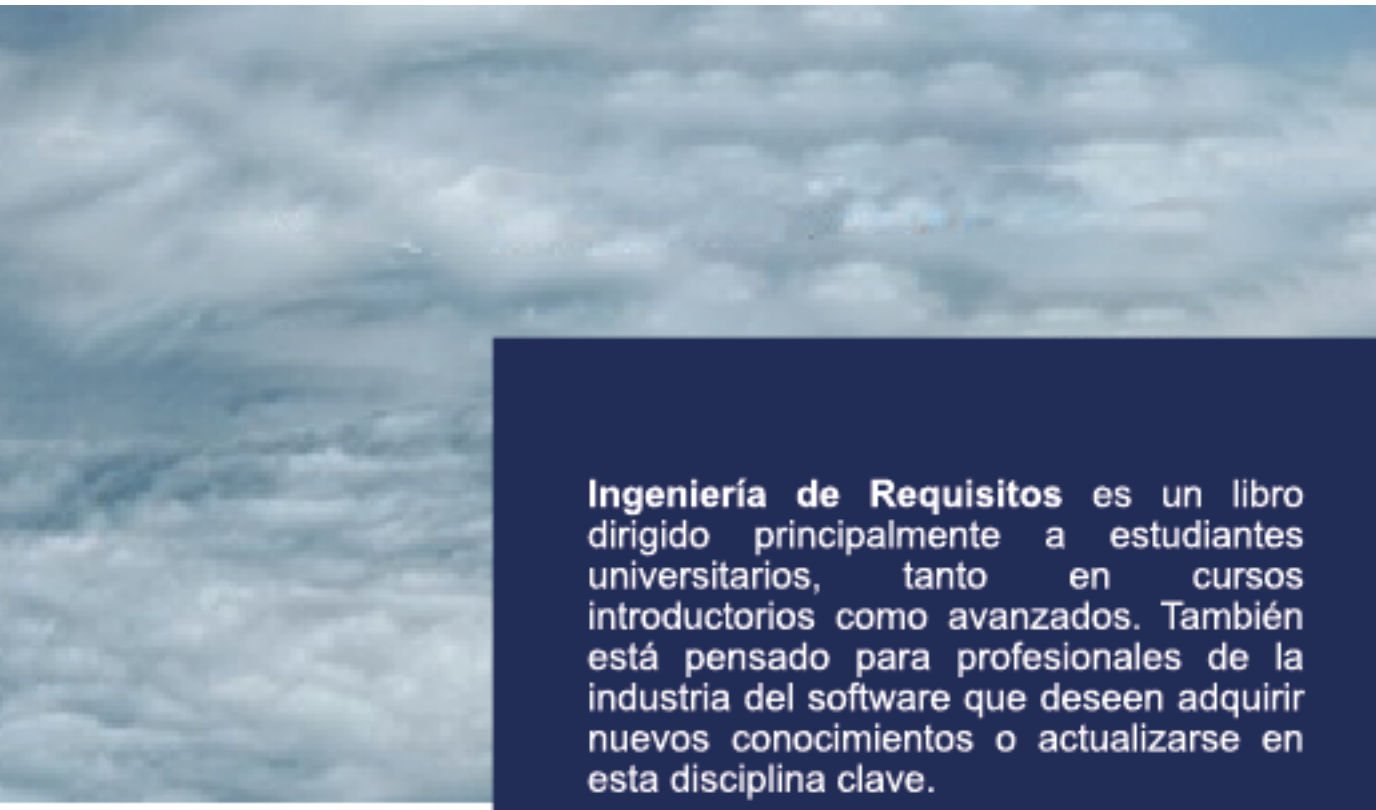
3.7 Requisitos de seguridad

ID: REQ-089
Descripción: El sistema debe contar con un mecanismo de seguridad para proteger las transacciones que requieren autorización o incluyen datos sensibles.
Fundamento: Garantizar la confidencialidad, integridad y autenticidad de las transacciones.
Origen: Regulaciones de seguridad de la información.
Dependencias: Cifrado de datos y autenticación de usuarios.
Dificultad: Alta.
Prioridad: Alta.

…

4. Anexos

4.1 Anexo A: Glosario LEL_R *(LEL de Requisitos)*
4.2 Anexo B: Escenarios futuros.
4.3 Anexo C: Mapa de relación entre escenarios futuros y requisitos (Pre-RS traceability).
4.4 Anexo D: Fichas de Información Extemporánea (FiE).
4.5 Anexo E: Esquemas de dependencias.



Ingeniería de Requisitos es un libro dirigido principalmente a estudiantes universitarios, tanto en cursos introductorios como avanzados. También está pensado para profesionales de la industria del software que deseen adquirir nuevos conocimientos o actualizarse en esta disciplina clave.

A lo largo de sus capítulos, la obra explora en profundidad la importancia de los requisitos en el desarrollo de software, desde sus fundamentos hasta estrategias efectivas para su gestión. Con un enfoque práctico y riguroso, se abordan temas esenciales como la incidencia de los requisitos en los proyectos software, la elicitación de información, el tratamiento de información extemporánea, los roles esenciales de la disciplina, el uso de modelos, la especificación de requisitos, la verificación y validación, la gestión de requisitos y el tratamiento de los requisitos en metodologías ágiles.

Este libro proporciona herramientas clave para definir requisitos de manera sólida, minimizando riesgos y maximizando el valor de las soluciones tecnológicas. Una guía imprescindible para establecer un punto de partida confiable y tomar el control del desarrollo de software de forma efectiva y eficiente.

DIIT
Departamento de Ingeniería e Investigaciones Tecnológicas

UNLaM
1989

Gladys Noemí Kaplan es Licenciada en Sistemas, Magister en Informática y Doctora en Ciencias Informáticas. Se desempeña como docente e investigadora en la Universidad Nacional de La Matanza desde el año 2005 donde, además, es responsable de la Línea de Investigación "Ingeniería de Requisitos: un enfoque a los negocios". También ejerce la docencia en carreras de grado y posgrado en diferentes Universidades Nacinales. Ha participado en numerosos eventos académicos y científicos tanto a nivel nacional como internacional publicando sus trabajos en diferentes libros (Perspectives on Software Requirements, Encyclopedia of Information Science & Technology) y revistas científicas (Requirements Engineering Journal, RITA, REDDI).

Firmantes

Universidad Nacional de La Matanza

Sistema: sudocu
Razón: Cargado por CECILIA GARGANO
Fecha de creación: 14/04/2025 11:50:05

Universidad Nacional de La Matanza

Sistema: sudocu
Razón: Autorizado por GABRIEL ESTEBAN BLANCO
Fecha de subida: 14/04/2025 19:28:08