Universidad Nacional Autónoma de México

Facultad de Ingeniería

Ingeniería de Software

Grupo 4

Ing. Orlando Zaldívar Zamorategui

Proyecto: Tutorial sobre fundamentos para los requerimientos del software. Definición de un requerimiento de software.

425490518 - Barrancos Parada Tiago
320304435 - Cabrera Rojas Oscar
322308381 - Calva Gatica Ángel David
319154780 - Esquivel Rosales Josselyn Anaid
320318467 - Estrada Martínez Mitzy Naomi

14 de abril de 2026 Ciudad Universitaria CDMX a 12 de mayo de 2026

Índice

1. Objetivo **1**

2. Introducción **1**

3. Marco Teórico **2**
3.1. Para empezar: ¿por qué deberíamos hablar de requerimientos funcionales? . . . 2
3.2. Una definición construida con muchas voces . . . 3
3.2.1. La frase central de Sommerville . . . 3
3.2.2. La misma idea, en español ecuatoriano . . . 3
3.2.3. La mirada industrial de Laplante . . . 3
3.2.4. La voz normativa de SWEBOK . . . 3
3.2.5. La definición compacta de Wiegers & Beatty . . . 4
3.2.6. El enfoque pragmático de Robertson & Robertson . . . 4
3.2.7. La voz colombiana en TIA . . . 4
3.2.8. El caso atípico de Pressman . . . 4
3.3. ¿De qué depende un requerimiento funcional? . . . 5
3.4. ¿En qué nivel se escribe un requerimiento funcional? . . . 5
3.4.1. El modelo binario clásico: usuario y sistema . . . 5
3.4.2. El modelo de tres niveles de Wiegers . . . 6
3.4.3. La jerarquía Volere de cinco niveles . . . 6
3.5. La frontera con los no funcionales . . . 6
3.5.1. ¿Qué son los requerimientos no funcionales? . . . 6
3.5.2. Una frontera difusa . . . 6
3.5.3. Los NFR como atributos sobre los FR . . . 7
3.5.4. Cuando los NFR generan FR derivados . . . 7
3.6. ¿Cuántos tipos de requerimientos hay? . . . 7
3.6.1. La taxonomía de tres tipos de Laplante . . . 7
3.6.2. La taxonomía de cinco tipos de Pressman . . . 7
3.7. De la teoría al papel: cómo se redacta un requerimiento funcional . . . 8
3.7.1. Lenguaje natural según el nivel . . . 8
3.7.2. Los cinco lineamientos de Sommerville . . . 8
3.7.3. La plantilla estructurada de siete campos . . . 8
3.7.4. Las reglas estrictas de Robertson & Robertson . . . 8
3.7.5. La estructura Description + Rationale . . . 9
3.7.6. El estándar IEEE 29148 según Laplante . . . 9
3.7.7. Cómo identificar y etiquetar un FR . . . 10
3.7.8. ¿Cómo se ve un FR bien escrito? . . . 10
3.8. Trazabilidad: cuando un FR es un nodo de una red . . . 10
3.8.1. Cuando un NFR genera un FR derivado . . . 10

4. Ejemplos **11**
4.1. Los ejemplos de Sommerville: el sistema MHC-PMS . . . 11
4.2. El ejemplo de la bomba de insulina aplicado a la plantilla de siete campos . . . 11
4.3. Los ejemplos de Laplante: tres casos de estudio recurrentes . . . 12
4.4. Los ejemplos de Robertson & Robertson: el caso IceBreaker . . . 12
4.5. Los ejemplos del Volere Knowledge Model . . . 13
4.6. Los ejemplos de Wiegers & Beatty: el sistema de aerolínea . . . 13

4.7. El ejemplo aislado de Pressman: el sistema de depósito con sensores 14
4.8. Los ejemplos de Celi-Párraga: el sistema de pedidos de alimentos 14
4.9. Los ejemplos de SWEBOK: capacidades elementales del software 14
4.10. Los ejemplos de Meyer et al.: la reclasificación crítica 15
4.11. Una mirada de conjunto a los ejemplos 15

5. Video **15**
5.1. Escaleta 16

6. Software interactivo **17**
6.1. Objetivo 17
6.2. Banco de preguntas rápidas 18

7. Cuestionario **20**

8. Conclusión **34**

9. Bibliografía **36**

1. Objetivo

El presente tutorial tiene como objetivo sentar las bases para lo que son y lo que siginifcan los requerimientos funcionales para el desarrollo de software, en específico, para la ingeneiría de software.

De tal forma que el lector podrá, al terminar el tutorial, gozar del entenedimiento completo sobre lo que son los requerimientos de software, los tipos que existen, de qué depende y cómo redactarlos, de manera teórica y práctica con los ejemplos mencionados.

El lector podrá evaluar su entendimiento por medio del cuestionario presentado.

Al concluir el tutorial, el lector deberá ser capaz de:

- Comprender qué es un requerimiento funcional, distinguirlo de un requerimiento no funcional e identificar los factores contextuales —tipo de software, perfil del usuario y enfoque organizacional— que condicionan su redacción.
- Distinguir los niveles de granularidad (usuario y sistema) y las taxonomías más utilizadas para clasificar los requerimientos a lo largo del ciclo de vida del software.
- Aplicar plantillas y reglas de estilo para escribir requerimientos funcionales claros, así como valorar su calidad mediante criterios como verificabilidad, no ambigüedad y testabilidad.
- Etiquetar requerimientos con identificadores únicos y reconocer su red de relaciones con reglas de negocio, casos de uso, diseño y código, entendiendo el papel de la trazabilidad.

2. Introducción

En la práctica de la Ingeniería de Software, pocas decisiones determinan tanto el éxito o fracaso de un proyecto como la manera en la que se definen sus requerimientos. Detrás de cada sistema que funciona bien hay un equipo que supo, en algún momento, responder con precisión a una pregunta aparentemente sencilla: ¿qué tiene que hacer este sistema? La respuesta, cuando se formaliza con rigor, da lugar a los requerimientos funcionales, y son ellos los que articulan el puente entre lo que el cliente necesita y lo que el equipo de desarrollo construye.

El presente trabajo se propone como un tutorial sobre la definición de requerimientos funcionales. La motivación es práctica: en cursos introductorios, en proyectos universitarios y en los primeros años de ejercicio profesional, la idea de «requerimiento funcional» suele presentarse de forma rápida, casi como un dato evidente, cuando en realidad es uno de los conceptos más ricos y debatidos del campo. Detrás de esa idea conviven al menos siete grandes formulaciones —las de Sommerville, Pressman, Laplante, Wiegers & Beatty, Robertson & Robertson, SWEBOK y Celi-Párraga et al.—, cada una con énfasis distintos: algunas insisten en los servicios que el sistema debe proveer, otras en los comportamientos que debe exhibir, otras en las acciones que el producto debe realizar para soportar el negocio del cliente. Comprender esas variaciones no es un ejercicio académico ocioso; es la diferencia entre redactar un requerimiento ambiguo —que se interpretará, casi inevitablemente, en favor del desarrollador y en contra del usuario— y redactar uno que sea, al mismo tiempo, comprensible, verificable y trazable.

El tutorial está pensado para estudiantes de ingeniería de software, pero también para profesionales que deseen sistematizar lo que saben sobre el tema. A lo largo del documento construimos una sola línea de pensamiento que va de lo más general a lo más concreto: partimos de la pregunta básica sobre qué es un requerimiento funcional, integramos las definiciones

más influyentes de la literatura, examinamos de qué factores depende su redacción, exploramos los niveles de granularidad en los que puede expresarse, delimitamos su frontera con los requerimientos no funcionales —frontera que, como veremos, no es tan nítida como suele presentarse—, revisamos las taxonomías que los autores proponen, y aterrizamos todo en la práctica concreta de la redacción, la identificación, la trazabilidad y la evaluación de calidad.

Tres ideas atraviesan el tutorial y conviene anunciarlas desde ya. La primera es que un requerimiento funcional, por muy bien definido que esté en abstracto, depende del contexto en el que se redacta: el tipo de software, el perfil del usuario y la cultura organizacional condicionan profundamente lo que se escribe y cómo se escribe. La segunda es que la testabilidad funciona como criterio operativo para distinguir un buen requerimiento funcional de uno mal planteado: si no es posible diseñar una prueba que lo verifique, el requerimiento está mal escrito, independientemente de cuán elegante suene. La tercera es que un requerimiento funcional no vive aislado, sino dentro de una red de relaciones —con requerimientos no funcionales, con reglas de negocio, con casos de uso, con elementos de diseño y de código—, y documentar esa red es lo que llamamos trazabilidad.

El documento privilegia un tono didáctico antes que enciclopédico. No buscamos agotar el campo, sino ofrecer una entrada coherente, cuidada y honesta a un concepto que el lector encontrará una y otra vez a lo largo de su carrera. Cada definición que se discute viene acompañada de la referencia exacta —libro, capítulo y página— que la sustenta, de modo que el lector interesado pueda regresar a la fuente y profundizar por su cuenta. Si al terminar la lectura logramos que se vea con más claridad qué es un requerimiento funcional, por qué importa redactarlo bien y cómo hacerlo, el tutorial habrá cumplido su propósito.

3. Marco Teórico

3.1. Para empezar: ¿por qué deberíamos hablar de requerimientos funcionales?

Cualquier proyecto de software se sostiene sobre lo que el sistema promete hacer. Los requerimientos son la base de todo proyecto de software, ya que describen las funcionalidades y características que el software debe tener para satisfacer las necesidades de los usuarios y/o clientes (Celi-Párraga et al., 2023, p. 3). Si esa base se construye con descuido, todo lo demás —arquitectura, diseño, código, pruebas— se erige sobre arena. Por eso la Ingeniería de Requerimientos dedica buena parte de su atención a clasificar, definir y comunicar lo que el sistema debe lograr.

Antes de avanzar conviene tener un mapa. La forma más extendida de organizar los requerimientos en el campo es la tipología dual: con frecuencia se mencionan dos grandes tipos de requerimientos, funcionales y no funcionales, aunque conviene tener cuidado con estas definiciones, puesto que no es raro encontrar que exista una relación entre ambas más cercana de lo que se espera (Contreras Bernal & Hernández Ruiz, 2022, p. 93). Esta advertencia no es menor: el campo carece de una definición precisa y consensuada de la distinción funcional/no funcional, y hay autores que incluso rechazan esa distinción mientras la mayor parte de la literatura la trata como un dato sin definirla con rigor (Meyer et al., 2019, §2, p. 2). La tensión teórica de fondo es real, y aprender a navegarla forma parte del oficio.

3.2. Una definición construida con muchas voces

3.2.1. La frase central de Sommerville

La definición canónica que casi todos los autores latinoamericanos recogen viene de Sommerville: los requerimientos funcionales para un sistema refieren lo que el sistema debe hacer (Sommerville, 2011, §4.1.1, p. 85). Esta brevedad esconde una formulación más extensa que conviene tener a mano: 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 (Sommerville, 2011, §4.1, p. 84).

Lo prohibido —lo que el sistema no debe hacer— es el detalle que muchas veces se pasa por alto, y que resulta crítico en sistemas críticos: pensemos en un sistema bancario que jamás debe permitir un saldo negativo no autorizado, o en un sistema médico que jamás debe administrar medicamentos sin doble verificación.

3.2.2. La misma idea, en español ecuatoriano

Celi-Párraga et al. (2023) reproducen casi palabra por palabra la formulación de Sommerville, lo que confirma que se trata de una definición consensuada en la tradición hispanoamericana. Los requerimientos funcionales, dicen, 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 (Celi-Párraga et al., 2023, §3.3, p. 39). La frase-resumen que ofrecen es exactamente la de Sommerville: los requerimientos funcionales para un sistema refieren lo que el sistema debe hacer (Celi-Párraga et al., 2023, §3.3, p. 39).

3.2.3. La mirada industrial de Laplante

Laplante (2017) aporta una formulación más industrial, pero igualmente compacta: los *functional requirements* (FRs) describen los servicios que el sistema debe proveer y cómo el sistema reaccionará ante sus entradas (Laplante, 2017, Cap. 1, p. 6)[1]. Y, con el mismo énfasis que Sommerville, añade que los requerimientos funcionales necesitan establecer explícitamente ciertos comportamientos que el sistema no debe ejecutar (Laplante, 2017, Cap. 1, p. 6)[2].

3.2.4. La voz normativa de SWEBOK

Bourque y Fairley (2014), como guía de cuerpo de conocimiento mantenida por la IEEE, ofrece la formulación más concisa y normativa: los requerimientos funcionales describen las funciones que el software debe ejecutar; por ejemplo, formatear un texto o modular una señal; a veces se les conoce como *capabilities* o *features* (Bourque & Fairley, 2014, Software Requirements KA, §1.3, p. 1-3). La verdadera novedad operativa que SWEBOK aporta no es la frase descriptiva, sino el criterio de identificación: un requerimiento funcional puede describirse como aquél para el cual puede escribirse un conjunto finito de pasos de prueba que validen su comportamiento (Bourque & Fairley, 2014, Software Requirements KA, §1.3, p. 1-3). Esto convierte la testabilidad en el sello distintivo: si no puede probarse, no es un requerimiento funcional bien planteado.

[1] Cita fuera del rango prioritario Laplante p.79–81 y p.93–97; se conserva por ser la definición canónica del libro.

[2] Cita fuera del rango prioritario; se conserva por completar la definición.

A esto se suma una propiedad más general: una propiedad esencial de todos los requerimientos de software es que sean verificables como característica individual en el caso de un requerimiento funcional, o a nivel del sistema en el caso de un requerimiento no funcional (Bourque & Fairley, 2014, Software Requirements KA, §1.1, p. 1-2).

3.2.5. La definición compacta de Wiegers & Beatty

Wiegers y Beatty (2013) ofrecen la formulación más breve y precisa de toda la literatura: un requerimiento funcional es una descripción de un comportamiento que un sistema exhibirá bajo condiciones específicas (Wiegers & Beatty, 2013, Tabla 1-1, p. 7). En doce palabras se cristalizan tres ideas: (i) se describe un comportamiento, no una función estática; (ii) ese comportamiento es exhibido —es decir, externamente observable— y por tanto verificable; (iii) bajo condiciones específicas, no en general.

La versión ampliada añade un matiz fundamental: los requerimientos funcionales especifican los comportamientos que el producto exhibirá bajo condiciones específicas, describiendo lo que los desarrolladores deben implementar para que los usuarios puedan cumplir sus tareas (requerimientos del usuario), satisfaciendo así los requerimientos del negocio; esta alineación entre los tres niveles de requerimientos es esencial para el éxito del proyecto (Wiegers & Beatty, 2013, p. 9).

3.2.6. El enfoque pragmático de Robertson & Robertson

Robertson y Robertson (2012) insisten en el aspecto pragmático: en esencia, un requerimiento funcional es algo que el producto debe hacer para soportar el negocio de su propietario (Robertson & Robertson, 2012, Cap. 1, Truth 10, p. 8)[3]. La definición canónica de apertura del Capítulo 10 es más operativa: los requerimientos funcionales especifican lo que el producto debe hacer: las acciones que debe realizar para satisfacer las razones fundamentales de su existencia (Robertson & Robertson, 2012, Cap. 10, p. 223).

3.2.7. La voz colombiana en TIA

Una formulación reciente, accesible y muy citable proviene del artículo de Contreras Bernal y Hernández Ruiz (2022), que define los requerimientos funcionales como aquellos que describen con detalle lo que el sistema debe hacer, es decir, su comportamiento a partir de indicaciones sobre cuáles son las excepciones, entradas y salidas del mismo; por lo anterior, dependen mucho del tipo de software, de los usuarios, así como de lo específico y particular que pueda ser el negocio de la organización (Contreras Bernal & Hernández Ruiz, 2022, p. 93)[4].

3.2.8. El caso atípico de Pressman

Pressman (2002) no proporciona una definición autocontenida del término. La formulación más cercana surge de su esquema de identificación: F = requisito funcional, D = requisito de datos, C = requisito de comportamiento, Z = requisito de interfaz, y S = requisito de salida; un requisito identificado como F09 indica que se trata de un requisito funcional y que tiene asignado el número 9 dentro de los citados requisitos (Pressman, 2002, §10.5.6, p. 175).

[3]Cita fuera del rango prioritario Robertson p.223–225, p.228–229, p.500; se conserva como apertura conceptual del libro.

[4]Cita en p.93, justo antes del rango prioritario TIA p.94–97; se conserva por ser la definición central del artículo.

La definición operativa de Pressman se construye más bien por contraste con la calidad: la calidad del software se define como concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados, y con las características implícitas que se espera de todo software desarrollado profesionalmente (Pressman, 2002, Cap. 8, p. 135).

3.3. ¿De qué depende un requerimiento funcional?

Hasta aquí tenemos siete formulaciones distintas pero compatibles. Lo importante es notar que ningún requerimiento funcional vive en el vacío. Sommerville (2011) lo deja claro: los requerimientos funcionales dependen del tipo de software que se esté desarrollando, de los usuarios esperados del software y del enfoque general que adopta la organización cuando se escriben los requerimientos (Sommerville, 2011, §4.1.1, p. 85). Esta misma formulación se reproduce literalmente en la obra ecuatoriana (Celi-Párraga et al., 2023, §3.3, p. 39), y se reafirma en el artículo colombiano que añade la dependencia respecto a lo específico y particular del negocio de la organización (Contreras Bernal & Hernández Ruiz, 2022, p. 93)[5].

Tres factores, entonces, condicionan lo que vamos a redactar: el dominio técnico, el perfil de usuario y la cultura organizacional. Un FR de un sistema embebido aeronáutico no se parece —ni puede parecerse— al de una aplicación de mensajería para adolescentes.

3.4. ¿En qué nivel se escribe un requerimiento funcional?

3.4.1. El modelo binario clásico: usuario y sistema

La estratificación más extendida en la literatura es la binaria. Sommerville (2011) la formula así: al expresarse como requerimientos del usuario, los requerimientos funcionales se describen por lo general de forma abstracta que entiendan los usuarios del sistema; sin embargo, requerimientos funcionales más específicos del sistema detallan las funciones del sistema, sus entradas y salidas, sus excepciones, etcétera (Sommerville, 2011, §4.1.1, p. 85). La misma distinción se reproduce en Celi-Párraga et al. (2023) (Celi-Párraga et al., 2023, §3.3, p. 39).

Para fijar la diferencia, conviene una formulación más amplia de Celi-Párraga: los requerimientos del usuario son enunciados, en un lenguaje natural junto con diagramas, acerca de qué servicios esperan los usuarios del sistema, y de las restricciones con las cuales éste debe operar; los requerimientos del sistema son descripciones más detalladas de las funciones, los servicios y las restricciones operacionales del sistema de software (Celi-Párraga et al., 2023, §3.2, pp. 37–38)[6].

Hay un rango interno, además: los requerimientos funcionales del sistema varían desde requerimientos generales que cubren lo que tiene que hacer el sistema, hasta requerimientos muy específicos que reflejan maneras locales de trabajar o los sistemas existentes de una organización (Sommerville, 2011, §4.1.1, p. 85). Laplante (2017) reproduce esta estratificación clásica: los requerimientos funcionales pueden ser de alto nivel y generales —en cuyo caso son requerimientos del usuario— o pueden ser detallados, expresando entradas, salidas, excepciones, etcétera —en cuyo caso son requerimientos del sistema (Laplante, 2017, Cap. 1, p. 6)[7].

[5] Cita en p.93, justo antes del rango prioritario TIA p.94–97; se conserva por reforzar el factor contextual.

[6] Cita fuera del rango prioritario Celi p.39–40; se conserva por aclarar la dicotomía usuario/sistema.

[7] Cita fuera del rango prioritario Laplante p.79–81 y p.93–97; se conserva por replicar la dicotomía clásica.

3.4.2. El modelo de tres niveles de Wiegers

Wiegers y Beatty (2013) introducen una distinción más sofisticada que se ha vuelto referencia industrial: los requerimientos del software incluyen tres niveles distintos: requerimientos de negocio, requerimientos del usuario y requerimientos funcionales; adicionalmente, todo sistema tiene una colección de requerimientos no funcionales (Wiegers & Beatty, 2013, p. 7). Una *feature* consiste en una o más capacidades del sistema lógicamente relacionadas que proveen valor a un usuario y son descritas por un conjunto de requerimientos funcionales (Wiegers & Beatty, 2013, p. 7).

La distinción de perspectiva resulta elegante: los requerimientos del usuario describen la vista del usuario; los requerimientos funcionales describen la vista del desarrollador (Wiegers & Beatty, 2013, p. 89)[8].

3.4.3. La jerarquía Volere de cinco niveles

Robertson y Robertson (2012) llevan la estratificación al extremo metodológico: trabajan con una jerarquía no subjetiva de cinco niveles —contexto de trabajo, evento de negocio, caso de uso de producto, paso del caso de uso de producto, y requerimiento atómico—. Los requerimientos funcionales bien redactados son requerimientos atómicos: ayuda a pensar en los requerimientos atómicos como el nivel más bajo de especificación de requerimientos (Robertson & Robertson, 2012, Cap. 10, p. 238)[9].

3.5. La frontera con los no funcionales

3.5.1. ¿Qué son los requerimientos no funcionales?

Para entender bien qué es un FR conviene mirar de frente a su contraparte. Celi-Párraga et al. (2023) la describen así: los requerimientos no funcionales son limitaciones sobre servicios o funciones que ofrece el sistema; incluyen restricciones tanto de temporización y del proceso de desarrollo, como impuestas por los estándares; los requerimientos no funcionales se suelen aplicar al sistema como un todo, más que a características o a servicios individuales del sistema (Celi-Párraga et al., 2023, §3.3, p. 40). Adicionalmente, los requerimientos no funcionales, como indica su nombre, son requerimientos que no se relacionan directamente con los servicios específicos que el sistema entrega a sus usuarios (Celi-Párraga et al., 2023, §3.3, p. 41)[10].

Los NFR a su vez se subdividen en tres grandes categorías: del producto (usabilidad, eficiencia, dependibilidad, seguridad), organizacionales (entorno, organizacionales, desarrollo) y externos (regulatorios, éticos, legislativos) (Celi-Párraga et al., 2023, §3.3, p. 41)[11].

3.5.2. Una frontera difusa

La distinción no es nítida. Sommerville (2011) lo advierte: si los requerimientos no funcionales se expresan por separado de los requerimientos funcionales, las relaciones entre ambos

[8]Cita fuera del rango prioritario Wiegers p.7–9 y p.208; se conserva por aclarar el contraste de perspectivas.

[9]Cita fuera del rango prioritario Robertson p.223–225, p.228–229, p.500; se conserva por presentar la jerarquía Volere completa.

[10]Cita en p.41, justo después del rango prioritario Celi p.39–40; se conserva por completar el contraste FR vs. NFR.

[11]Cita en p.41, justo después del rango prioritario Celi p.39–40; se conserva por completar la taxonomía de NFR.

serían difíciles de entender (Sommerville, 2011, Cap. 4, p. 90)[12].

3.5.3. Los NFR como atributos sobre los FR

Bourque y Fairley (2014) establece la relación estructural más clara que ofrece la literatura: los requerimientos de calidad del software son en realidad atributos de (o restricciones sobre) los requerimientos funcionales (lo que el sistema hace); este KA busca claridad usando software quality en el sentido más amplio y usando los requerimientos de calidad del software como restricciones sobre los requerimientos funcionales (Bourque & Fairley, 2014, Software Quality KA, p. 10-1).

Y de manera complementaria: los requerimientos no funcionales son aquellos que actúan para restringir la solución; a veces se les conoce como *constraints* o *quality requirements*; pueden clasificarse adicionalmente en requerimientos de rendimiento, mantenibilidad, seguridad (safety), confiabilidad, seguridad (security), interoperabilidad u otros (Bourque & Fairley, 2014, Software Requirements KA, §1.3, p. 1-3).

Una idea para llevarse a casa: el FR es el contenedor («qué hace el sistema»); el NFR es el calificativo («qué tan bien lo hace»). No son hermanos paralelos, sino padre e hijo conceptuales.

3.5.4. Cuando los NFR generan FR derivados

Un NFR puede derivar en uno o varios FR concretos. Los NFR pueden asociarse a los requerimientos funcionales, por ejemplo: «solo usuarios autorizados deberían poder acceder a la funcionalidad X del sistema» (Laplante, 2017, p. 10)[13]. Wiegers y Beatty (2013) dan el ejemplo paradigmático: los requerimientos de seguridad para autenticación de usuarios dan lugar a requerimientos funcionales derivados que podrían implementarse a través de funcionalidad de contraseñas o biometría (Wiegers & Beatty, 2013, p. 290)[14].

3.6. ¿Cuántos tipos de requerimientos hay?

3.6.1. La taxonomía de tres tipos de Laplante

Laplante (2017) amplía la dicotomía clásica con una tercera categoría: otra taxonomía para las especificaciones de requerimientos se enfoca en el tipo de requerimiento de la siguiente lista de posibilidades: requerimientos funcionales (FRs), requerimientos no funcionales (NFRs) y requerimientos de dominio (Laplante, 2017, Cap. 1, p. 6)[15]. Los requerimientos de dominio son derivados del dominio de aplicación; estos tipos de requerimientos pueden consistir en nuevos requerimientos funcionales o restricciones a requerimientos funcionales existentes, o pueden especificar cómo deben realizarse cómputos particulares (Laplante, 2017, p. 11)[16].

3.6.2. La taxonomía de cinco tipos de Pressman

Pressman (2002), en cambio, propone una taxonomía con más categorías: el tipo de requisito toma alguno de los siguientes valores: F = requisito funcional, D = requisito de datos, C = re-

[12]Cita en p.90, justo después del rango prioritario Sommerville p.84–86 y p.98; se conserva por documentar la frontera difusa.

[13]Cita fuera del rango prioritario Laplante p.79–81 y p.93–97; se conserva por documentar la asociación NFR→FR.

[14]Cita fuera del rango prioritario Wiegers p.7–9 y p.208; se conserva por su valor pedagógico.

[15]Cita fuera del rango prioritario Laplante p.79–81 y p.93–97; se conserva por presentar la taxonomía base del libro.

[16]Cita fuera del rango prioritario Laplante; se conserva por completar la taxonomía de tres tipos.

quisito de comportamiento, Z = requisito de interfaz, y S = requisito de salida (Pressman, 2002, §10.5.6, p. 175).

3.7. De la teoría al papel: cómo se redacta un requerimiento funcional

Aquí entra la parte práctica del tutorial. Tener clara la definición es necesario pero insuficiente: hay que saber escribirla.

3.7.1. Lenguaje natural según el nivel

Celi-Párraga et al. (2023) marcan el principio rector: los requerimientos del usuario para un sistema deben describir los requerimientos funcionales y no funcionales, de forma que sean comprensibles para los usuarios del sistema que no cuentan con un conocimiento técnico detallado; el documento de requerimientos no debe incluir detalles de la arquitectura o el diseño del sistema; en consecuencia, si se escriben los requerimientos del usuario, no se debe usar jerga de software, anotaciones estructuradas o formales; deben escribirse los requerimientos del usuario en lenguaje natural, con tablas y formas sencillas, así como diagramas intuitivos (Celi-Párraga et al., 2023, §3.5, p. 44)[17].

A nivel del sistema cambian las herramientas: los requerimientos del usuario se escriben casi siempre en lenguaje natural, complementado con diagramas y tablas adecuados en el documento de requerimientos; los requerimientos del sistema se escriben también en lenguaje natural, pero de igual modo se utilizan otras notaciones basadas en formas, modelos gráficos del sistema o modelos matemáticos del sistema (Celi-Párraga et al., 2023, §3.5, p. 44)[18].

3.7.2. Los cinco lineamientos de Sommerville

Para minimizar la interpretación errónea al escribir los requerimientos en lenguaje natural, Sommerville (2011) recomienda (Sommerville, 2011, p. 96)[19]: (i) formato estándar uniforme para todos los requerimientos —un requerimiento, una oración— asociado con su razón y fuente; (ii) distinguir obligatorio de deseable mediante verbos consistentes («debe» vs. «debería»); (iii) resaltar las partes clave con negrilla, cursiva o color; (iv) evitar la jerga técnica; (v) asociar una razón a cada requerimiento de usuario para facilitar el cambio futuro.

3.7.3. La plantilla estructurada de siete campos

Cuando el lenguaje natural se queda corto, Sommerville (2011) ofrece una plantilla con siete campos: cuando se use una forma estándar para especificar requerimientos funcionales, se debe incluir la siguiente información (Sommerville, 2011, §4.3.2, p. 98): (1) función o entidad —descripción de qué se especifica; (2) entradas y procedencias; (3) salidas y destinos; (4) requiere —datos u otras entidades del sistema necesarios para el cálculo; (5) acción —qué se va a hacer; (6) precondición y postcondición; (7) efectos colaterales.

3.7.4. Las reglas estrictas de Robertson & Robertson

Robertson y Robertson (2012) llevan la disciplina de redacción al extremo. El nivel de detalle de sus ejemplos es revelador: los requerimientos están escritos como una única oración con un

[17] Cita fuera del rango prioritario Celi p.39–40; se conserva por su valor práctico para la redacción.
[18] Cita fuera del rango prioritario Celi p.39–40; se conserva por su valor práctico.
[19] Cita en p.96, fuera del rango prioritario Sommerville p.84–86 y p.98; se conserva por su valor práctico.

único verbo; cuando se escribe esta única oración, se hace el requerimiento mucho más fácil de probar y mucho menos susceptible a la ambigüedad (Robertson & Robertson, 2012, Cap. 10, p. 228). La forma canónica es «The product shall...»: hace la oración activa y se enfoca en comunicar lo que el producto pretende hacer (Robertson & Robertson, 2012, p. 228).

Hay un anti-patrón frecuente: a veces las personas usan una mezcla de «shall», «must», «will», «might», «could», etc., para indicar la prioridad de un requerimiento; esta práctica resulta en confusión semántica y se aconseja contra ella; en su lugar, debe usarse una forma consistente para escribir las descripciones de los requerimientos —«The product shall...» es la más común— y usar un atributo separado del requerimiento para identificar su prioridad (Robertson & Robertson, 2012, p. 229).

3.7.5. La estructura Description + Rationale

Una de las contribuciones más originales de Robertson y Robertson (2012) al campo es exigir que cada requerimiento lleve una explicación de por qué existe. Se sugiere —fuertemente— que se añada un *rationale* a los requerimientos para mostrar por qué existe el requerimiento; en algunos casos puede ser obvio, pero en muchas circunstancias es un componente crucial del requerimiento (Robertson & Robertson, 2012, p. 229).

Un ejemplo concreto: Description: «The product shall record roads that have been treated.» Rationale: «To be able to schedule untreated roads and highlight potential danger.» (Robertson & Robertson, 2012, pp. 229–230).

¿Por qué importa el rationale? Al incluir un rationale con la descripción, el requerimiento mismo se vuelve más útil; al saber por qué algo está ahí, los desarrolladores y los testers saben mucho más sobre el esfuerzo que deben dedicarle; también indica a los mantenedores futuros por qué existe el requerimiento en primer lugar (Robertson & Robertson, 2012, p. 230)[20]. Y como bono: el rationale también ayuda a superar la posibilidad de escribir inadvertidamente una solución en lugar de la necesidad real (Robertson & Robertson, 2012, p. 230)[21].

3.7.6. El estándar IEEE 29148 según Laplante

Laplante (2017) aporta una regla operativa que ningún otro autor enuncia con tanta precisión: los requerimientos funcionales deben capturar todas las entradas del sistema y la secuencia exacta de operaciones y respuestas (salidas) ante situaciones normales y anormales para cada posibilidad de entrada (Laplante, 2017, Cap. 4, p. 98)[22]. Adicionalmente, los requerimientos funcionales pueden usar descripción caso-por-caso u otras formas generales de descripción (Laplante, 2017, p. 98)[23].

IEEE 29148 provee un número de opciones organizacionales para los requerimientos funcionales; los requerimientos funcionales pueden organizarse por: modo del sistema, clase de usuario, objeto, característica, estímulo, jerarquía funcional (Laplante, 2017, p. 99)[24]. Una combinación de estas técnicas puede usarse en un mismo documento SRS (Laplante, 2017, p. 99)[25].

[20]Cita en p.230, justo después del rango prioritario Robertson p.228–229; se conserva como cierre del argumento del rationale.

[21]Cita en p.230, justo después del rango prioritario Robertson; se conserva como salvaguarda anti-pseudo-requerimiento.

[22]Cita en p.98, justo después del rango prioritario Laplante p.93–97; se conserva como regla central de redacción.

[23]Cita en p.98, justo después del rango prioritario Laplante; se conserva por reforzar la regla anterior.

[24]Cita en p.99, justo después del rango prioritario Laplante p.93–97; se conserva por documentar las opciones organizacionales.

[25]Cita en p.99, justo después del rango prioritario Laplante; se conserva por completar el principio.

3.7.7. Cómo identificar y etiquetar un FR

La identificación trazable es central. Pressman (2002) propone que un requisito identificado como F09 indica que se trata de un requisito funcional y que tiene asignado el número 9 dentro de los citados requisitos (Pressman, 2002, §10.5.6, p. 175).

Wiegers y Beatty (2013) ofrecen una convención más rica: el enfoque más simple da a cada requerimiento un número de secuencia único, como UC-9 o FR-26; el prefijo indica el tipo de requerimiento, como FR para requerimiento funcional (Wiegers & Beatty, 2013, p. 187)[26]. Y para cuando el FR es jerárquico: una buena convención es escribir el requerimiento padre para que parezca un título, encabezado o nombre de feature, en lugar de parecerse a un requerimiento funcional en sí mismo; los requerimientos hijos de ese padre, en agregado, entregan la capacidad descrita en el padre (Wiegers & Beatty, 2013, p. 188)[27].

3.7.8. ¿Cómo se ve un FR bien escrito?

¿Cómo saber si un FR está bien escrito? Wiegers y Beatty (2013) enumeran los atributos individuales de un FR excelente (Wiegers & Beatty, 2013, Cap. 11, pp. 204–205)[28]: completitud (*Complete*), corrección (*Correct*), factibilidad (*Feasible*), necesidad (*Necessary*), priorización (*Prioritized*), no ambigüedad (*Unambiguous*) y verificabilidad (*Verifiable*). Y los atributos del conjunto de FRs (Wiegers & Beatty, 2013, p. 206)[29]: completitud (no faltan FRs), consistencia (no se contradicen entre sí), modificabilidad (se pueden modificar manteniendo histórico) y trazabilidad (bidireccional).

3.8. Trazabilidad: cuando un FR es un nodo de una red

Un requerimiento funcional no es un átomo aislado: vive enredado en una red de orígenes y consecuencias. Documentar esa red es la trazabilidad.

3.8.1. Cuando un NFR genera un FR derivado

Wiegers y Beatty (2013) ofrecen el ejemplo paradigmático: los requerimientos de seguridad para autenticación de usuarios dan lugar a requerimientos funcionales derivados que podrían implementarse a través de funcionalidad de contraseñas o biometría; en esos casos, se pueden rastrear los requerimientos funcionales correspondientes hacia atrás a su requerimiento no funcional padre y hacia adelante a los entregables aguas abajo como de costumbre (Wiegers & Beatty, 2013, Cap. 29, p. 497)[30].

Cuando llega la hora de gestionar el conjunto de FR de un proyecto, Laplante (2017) retoma las preguntas de Andriole (Laplante, 2017, Cap. 9, p. 223)[31]: (a) ¿cuáles son las cosas específicas que el sistema debe hacer para satisfacer los requerimientos propositivos? (b) ¿cómo deberían ser priorizados? (c) ¿cuáles son los riesgos de implementación? Es una lista mínima útil para revisión técnica.

[26] Cita en p.187, fuera del rango prioritario Wiegers p.7–9 y p.208; se conserva por documentar las convenciones de etiquetado.

[27] Cita en p.188, fuera del rango prioritario Wiegers; se conserva por documentar las convenciones jerárquicas.

[28] Cita fuera del rango prioritario Wiegers p.7–9 y p.208; se conserva por documentar los atributos de calidad.

[29] Cita fuera del rango prioritario Wiegers; se conserva por completar los atributos del conjunto.

[30] Cita fuera del rango prioritario Wiegers p.7–9 y p.208; se conserva por documentar la trazabilidad NFR→FR.

[31] Cita fuera del rango prioritario Laplante p.79–81 y p.93–97; se conserva por documentar la lista de Andriole.

4. Ejemplos

Las definiciones formales y las plantillas adquieren su sentido pleno cuando se aplican a casos concretos. Esta sección recoge los ejemplos de requerimientos funcionales que aparecen en la literatura analizada, agrupados por autor y por sistema de estudio. Conservamos, donde es posible, la formulación original con la que cada autor presenta su FR, porque ahí está la lección: no son enunciados inventados para ilustrar una idea, sino casos de proyectos reales o de referencia que cada autor ha utilizado para enseñar a redactar.

4.1. Los ejemplos de Sommerville: el sistema MHC-PMS

Sommerville (2011) construye casi todo el Capítulo 4 alrededor de un caso de estudio: el *Mental Health Care Patient Management System* (MHC-PMS), un sistema de gestión de pacientes de salud mental. El ejemplo es valioso porque muestra cómo un único requerimiento del usuario, redactado en una línea, se descompone en varios requerimientos del sistema mucho más detallados.

A nivel de *requerimiento del usuario*, el enunciado es breve: el MHC-PMS elaborará mensualmente informes administrativos que revelen el costo de los medicamentos prescritos por cada clínica durante ese mes (Sommerville, 2011, §4.1.1, p. 85). Esa misma necesidad, expresada como *requerimientos del sistema*, se descompone en cinco sub-requerimientos numerados que cubren detalles operativos como el día de generación, la impresión automática después de las 17:30, el formato del reporte por clínica, el manejo de unidades de dosis distintas y la restricción de acceso a usuarios autorizados (Sommerville, 2011, §4.1.1, p. 85).

Otro ejemplo emblemático del libro es el del requerimiento ambiguo: «Un usuario podrá buscar en todas las clínicas las listas de citas». La frase parece simple, pero esconde una trampa de interpretación: mientras el personal médico esperaba que el sistema buscara automáticamente en todas las clínicas dado el nombre de un paciente, un desarrollador podría implementarlo exigiendo que el usuario eligiera primero una clínica y luego buscara dentro de ella. Como advierte el autor, es natural que un desarrollador de sistemas interprete un requerimiento ambiguo de forma que simplifique su implementación (Sommerville, 2011, Cap. 4, p. 86). La lección es directa: la ambigüedad nunca se resuelve a favor del usuario, sino del desarrollador.

4.2. El ejemplo de la bomba de insulina aplicado a la plantilla de siete campos

El segundo gran caso de estudio que recorre el libro de Sommerville (2011) es el sistema de control de una *bomba de insulina*. Aquí el autor aplica los cinco lineamientos de redacción en lenguaje natural, ofreciendo un FR como: «Si se requiere, cada 10 minutos el sistema medirá el azúcar en la sangre y administrará insulina», acompañado de la razón explícita entre paréntesis —los cambios de azúcar en sangre son relativamente lentos, por lo que mediciones más frecuentes serían innecesarias y mediciones menos periódicas podrían conducir a niveles de azúcar elevados— (Sommerville, 2011, p. 96)[32].

El mismo sistema reaparece en la plantilla estructurada de siete campos: la *Función* es calcular la dosis de insulina cuando el azúcar actual está entre 3 y 7 unidades; las *Entradas* son la lectura actual del sensor y dos lecturas previas; la *Salida* es `CompDose`, la dosis que se administrará al ciclo de control principal; la *Acción* calcula `CompDose` en función de la

[32]Cita en p. 96, fuera del rango prioritario Sommerville p.84–86 y p.98; se conserva por ser el ejemplo aplicado de los cinco lineamientos.

pendiente del nivel de azúcar; la *Precondición* exige que el depósito contenga al menos una dosis máxima; la *Postcondición* actualiza las lecturas previas; y los *Efectos colaterales* son inexistentes (Sommerville, 2011, §4.3.2, p. 98). Todo ese aparato no es burocrático: elimina ambigüedad, fuerza a pensar en pre y postcondiciones, y entrega al equipo de pruebas una base verificable.

4.3. Los ejemplos de Laplante: tres casos de estudio recurrentes

Laplante (2017) ancla casi cada concepto del libro en tres casos de estudio: un sistema de manejo de equipajes (*baggage handling system*), un punto de venta para una tienda de mascotas (*pet store POS system*) y un hogar inteligente (*smart home system*).

Para el sistema de manejo de equipaje, el autor ofrece una muestra que ilustra el uso del verbo «shall» y de la numeración jerárquica: «El sistema deberá manejar hasta 20 equipajes por minuto», y, como ejemplo de comportamiento prohibido, «cuando el sistema esté en modo inactivo, la banda transportadora no deberá moverse» (Laplante, 2017, Cap. 1, p. 7)[33].

Para el sistema de punto de venta, un FR ilustrativo es: «Cuando el operador presione el botón "total", la venta actual entrará en estado cerrado» (Laplante, 2017, Cap. 1, p. 7)[34]. Nótese cómo el enunciado describe simultáneamente una entrada (la pulsación del botón), una transición de estado y la respuesta esperada del sistema, en una sola oración.

Laplante (2017) también utiliza estos casos para mostrar la asociación entre requerimientos no funcionales y funcionales. Un NFR de seguridad típico, formulado como «sólo los usuarios autorizados deberían poder acceder a la funcionalidad X del sistema», se traduce en uno o más FRs concretos que implementen la autenticación y el control de acceso (Laplante, 2017, p. 10)[35].

4.4. Los ejemplos de Robertson & Robertson: el caso IceBreaker

Robertson y Robertson (2012) construyen toda la metodología Volere alrededor del proyecto IceBreaker, un sistema de planeación del deshielo de carreteras. El ejemplo introductorio, que aparece muy temprano en el libro, ya muestra la forma canónica «The product shall...» en acción: «The product shall produce a schedule of all roads upon which ice is predicted to form within the given time parameter» (Robertson & Robertson, 2012, Cap. 1, p. 10)[36]. La definición canónica del Capítulo 10 se ilustra con un ejemplo análogo: «The product shall predict which road sections will freeze within the selected time parameters» (Robertson & Robertson, 2012, Cap. 10, p. 223).

El verdadero valor pedagógico aparece, sin embargo, en la descomposición de un escenario de caso de uso. Partiendo del paso «El ingeniero proporciona una fecha de programación y un identificador de distrito», los autores derivan cinco FRs atómicos: (i) «The product shall accept a scheduling date»; (ii) «The product shall warn if the scheduling date is neither today nor the next day»; (iii) «The product shall accept a valid district identifier»; (iv) «The product shall verify that the district is within the de-icing responsibility of the area covered by this installation»; (v) «The product shall verify that the district is the one wanted by the engineer»

[33]Cita en p. 7, fuera del rango prioritario Laplante p.79–81 y p.93–97; se conserva por ser el ejemplo canónico del baggage handling system.

[34]Cita en p. 7, fuera del rango prioritario Laplante p.79–81 y p.93–97; se conserva por ser el ejemplo canónico del pet store POS system.

[35]Cita en p. 10, fuera del rango prioritario Laplante p.79–81 y p.93–97; se conserva por ilustrar la traducción NFR→FR.

[36]Cita en p. 10, fuera del rango prioritario Robertson p.223–225, p.228–229, p.500; se conserva por ser el primer ejemplo IceBreaker del libro.

(Robertson & Robertson, 2012, Cap. 10, p. 227)[37]. Cinco oraciones, cinco verbos activos, cinco cosas verificables. Es la disciplina Volere en acción.

El ejemplo más recordado del libro es el del par *Description + Rationale* aplicado al mismo proyecto: *Description*: «The product shall record roads that have been treated»; *Rationale*: «To be able to schedule untreated roads and highlight potential danger» (Robertson & Robertson, 2012, pp. 229–230). El rationale, en este caso, eleva un FR aparentemente trivial a la categoría de requerimiento crítico: de él dependen vidas humanas en carreteras heladas.

4.5. Los ejemplos del Volere Knowledge Model

Cuando Robertson y Robertson (2012) formalizan la clase *Functional Requirement* en el Apéndice D del libro, ilustran el concepto con cuatro ejemplos cortos pero altamente representativos: «calcular la tarifa», «analizar la composición química», «registrar el cambio de nombre», «encontrar la nueva ruta» (Robertson & Robertson, 2012, Apéndice D, p. 500). La elección no es casual: cada uno de los cuatro verbos —calcular, analizar, registrar, encontrar— corresponde a una de las operaciones CRUD (*create*, *read*, *update*, *delete*) que los autores consideran consustanciales a todo FR.

4.6. Los ejemplos de Wiegers & Beatty: el sistema de aerolínea

Wiegers y Beatty (2013) acompañan su definición del FR como «descripción de un comportamiento que un sistema exhibirá bajo condiciones específicas» con dos ejemplos icónicos del sistema de reservas de una aerolínea, ambos redactados con la fórmula «shall»: «El Pasajero deberá poder imprimir tarjetas de embarque para todos los segmentos de vuelo en los que ha hecho check-in», y «Si el perfil del Pasajero no indica preferencia de asiento, el sistema de reservas deberá asignar un asiento» (Wiegers & Beatty, 2013, p. 9).

El primero ilustra la plantilla Alexander-Stevens —«El [actor] deberá poder [hacer algo] [a un objeto] [con condiciones]»—; el segundo ilustra la sintaxis EARS para FRs condicionales —«Si [precondición], el sistema deberá [respuesta]»— (Wiegers & Beatty, 2013, p. 208).

El otro caso de estudio recurrente del libro es el *Chemical Tracking System* (CTS), del que los autores extraen un ejemplo aplicado a la plantilla Alexander-Stevens: «El Químico deberá poder reordenar cualquier producto químico que haya pedido en el pasado mediante la recuperación y edición de los detalles del pedido» (Wiegers & Beatty, 2013, p. 208). Adicionalmente, cuando el FR cae naturalmente del diálogo entre actor y sistema en un caso de uso, los autores muestran un ejemplo prototípico: «El sistema deberá asignar un número de secuencia único a cada solicitud» (Wiegers & Beatty, 2013, Cap. 8, p. 162)[38].

Para el SRS de muestra del CTS en el Apéndice C, Wiegers y Beatty (2013) emplean identificadores jerárquicos textuales como `Request.Find.Vendor` o `Request.Find.HazRefs`, en lugar de números secuenciales como `FR-26` (Wiegers & Beatty, 2013, p. 188)[39]. La intención es que el identificador hable por sí mismo: leer la etiqueta basta para entender de qué FR se trata, sin tener que cruzarla contra una tabla.

[37]Cita en p. 227, justo dentro del rango prioritario Robertson p.223–225 y adyacente al rango p.228–229; se conserva por ser la aplicación práctica del método de derivación.

[38]Cita en p. 162, fuera del rango prioritario Wiegers p.7–9 y p.208; se conserva por ilustrar la derivación natural de FRs desde casos de uso.

[39]Cita en p. 188, fuera del rango prioritario Wiegers p.7–9 y p.208; se conserva por ser el ejemplo de la convención de etiquetado jerárquico.

4.7. El ejemplo aislado de Pressman: el sistema de depósito con sensores

Pressman (2002) no abunda en ejemplos ilustrativos. De hecho, en todo el libro aparece un único ejemplo concreto de cómo redactar un FR en lenguaje natural, situado en el capítulo dedicado a métodos formales: «El sistema debe de mantener el nivel horario del depósito a partir de los sensores de profundidad situados en el depósito. Estos valores deben de ser almacenados para los últimos seis meses» (Pressman, 2002, Cap. 25, p. 437)[40].

El ejemplo es valioso por su economía: en dos oraciones, el autor identifica el sujeto («el sistema»), un verbo modal de obligación («debe de»), las acciones específicas (mantener, almacenar), las condiciones cuantificables (horario, seis meses) y el origen de los datos (sensores de profundidad). Es prácticamente la plantilla Alexander-Stevens implícita, redactada en español castizo.

En cuanto a la identificación, Pressman (2002) propone un esquema mínimo: el código F seguido de un número, donde un requisito etiquetado como `F09` indica que se trata de un requisito funcional y que le ha sido asignado el número 9 dentro de los citados requisitos (Pressman, 2002, §10.5.6, p. 175). La convención coincide con la recomendación de Wiegers y Beatty (2013) de usar el prefijo `FR` seguido de un número de secuencia único, como `FR-26` o `UC-9` (Wiegers & Beatty, 2013, p. 187)[41].

4.8. Los ejemplos de Celi-Párraga: el sistema de pedidos de alimentos

Celi-Párraga et al. (2023) introducen un ejemplo gráfico que sirve para ilustrar simultáneamente los conceptos de actor, interacción y FR. El sistema de pedidos de alimentos se modela con cuatro actores —Cliente, Vendedor, Administrador, Cocinero— y siete interacciones funcionales identificadas: pedir los alimentos, entregar dinero, realizar los alimentos, entregar los alimentos, calentar los alimentos, pago de los productos, y la relación con proveedores y ganancias (Celi-Párraga et al., 2023, §3.3, p. 40). Cada una de esas interacciones constituye un requerimiento funcional candidato del sistema, y el diagrama deja ver cómo los FRs no son simples enunciados aislados, sino nodos en una red de relaciones entre quienes usan el sistema y los servicios que éste provee.

4.9. Los ejemplos de SWEBOK: capacidades elementales del software

Bourque y Fairley (2014) ilustran su definición de FR con dos casos deliberadamente sencillos: «formatear un texto» y «modular una señal» (Bourque & Fairley, 2014, Software Requirements KA, §1.3, p. 1-3). La elección es intencional. Al elegir ejemplos de tan distinta naturaleza —uno es una operación de software ofimático, el otro una operación de procesamiento de señales—, los autores subrayan que la categoría «requerimiento funcional» es transversal a todos los dominios de aplicación. Lo que hace funcional a un requerimiento no es el dominio en el que vive, sino el hecho de describir una función que el software debe ejecutar.

[40] Cita en p. 437, fuera del rango prioritario Pressman p.135–137 y p.175–177; se conserva por ser el único ejemplo concreto de redacción de FR en todo el libro.

[41] Cita en p. 187, fuera del rango prioritario Wiegers p.7–9 y p.208; se conserva por documentar la convención de etiquetado paralela a la de Pressman.

4.10. Los ejemplos de Meyer et al.: la reclasificación crítica

Meyer et al. (2019) adoptan un enfoque distinto: en lugar de proponer sus propios ejemplos, toman enunciados típicos de documentos industriales y los reclasifican bajo su taxonomía. El ejemplo prototípico de la categoría *Behavior* —que ellos consideran la categoría matriz dentro de la cual se inscribe lo «funcional»— es deliberadamente mínimo: «Mostrar la lista de elementos disponibles» (Meyer et al., 2019, §4.1, p. 7). Es un comportamiento que especifica un efecto observable —qué muestra el sistema— sin determinar el cómo, y por eso califica como propiedad funcional.

Los ejemplos de timing y seguridad —«límites de tiempo» y «condiciones de seguridad»— se citan, en cambio, como casos clásicos de comportamiento no funcional, esto es, propiedades sobre cómo se logran los efectos del sistema (Meyer et al., 2019, §4.2, p. 9).

El ejercicio se vuelve más interesante cuando los autores aplican la taxonomía a un documento real (Bair, 2006). Un enunciado típicamente catalogado como funcional —«El sistema deberá permitir el pedido de productos en línea por parte del cliente o del agente de ventas»— se reclasifica simplemente como *Behavior* (Meyer et al., 2019, §6, p. 15)[42]. En contraste, un enunciado típicamente catalogado como no funcional —«El sistema deberá estar completamente operativo al menos el x % del tiempo»— se reclasifica como *Constraint* (Meyer et al., 2019, §6, p. 16)[43]. La lección es práctica: distinga siempre entre lo que el sistema hace (Behavior, FR) y las restricciones sobre cómo lo hace o cuándo está disponible (Constraint, NFR).

4.11. Una mirada de conjunto a los ejemplos

Si se observan en bloque, los ejemplos comparten patrones que merecen señalarse. Casi todos comienzan con un sujeto explícito —«el sistema», «el producto», «el Pasajero», «el operador»—; casi todos usan un verbo modal de obligación —«deberá», «debe», «shall», «must»— en una sola oración con un solo verbo principal; casi todos cierran con condiciones cuantificables o restricciones explícitas. Cuando un autor se aparta de este patrón —Pressman al ofrecer un único ejemplo en todo el libro, Meyer al renunciar al verbo modal y favorecer la categoría Behavior—, lo hace por razones argumentativas claras y conscientes.

La diversidad de dominios es también notable: salud mental, dispositivos

médicos, equipajes, comercio minorista, hogares inteligentes, deshielo de carreteras, aerolíneas, química industrial, software ofimático, procesamiento de señales, pedidos de alimentos. Esta variedad confirma la afirmación normativa de SWEBOK: un FR no se define por el dominio en el que vive, sino por la posibilidad de que un conjunto finito de pruebas valide su comportamiento.

5. Video

El siguiente video fue producido por el equipo como recurso de apoyo para la comprensión de los fundamentos de los requerimientos de software. A lo largo de aproximadamente cuatro minutos, la presentadora introduce el concepto de requerimiento de software, distingue entre requerimientos funcionales y no funcionales, ilustra ambos tipos con un ejemplo concreto de una aplicación bancaria móvil, describe el ciclo básico de obtención, análisis, documentación y

[42]Cita en §6, fuera del rango prioritario Meyer (primeras páginas hasta la primera tabla); se conserva por ilustrar el ejercicio de reclasificación aplicado.

[43]Cita en §6, fuera del rango prioritario Meyer (primeras páginas hasta la primera tabla); se conserva por completar el ejercicio de reclasificación.

validación de requerimientos, señala los errores más comunes en su definición y cierra con las buenas prácticas que permiten evitarlos. La siguiente escaleta detalla los segmentos del video con sus respectivos tiempos y locuciones tal como fueron grabados.

5.1. Escaleta

00:00	Hola, mi nombre es Mitzi y soy parte del equipo 5 de la materia de Ingeniería en Software.
00:04	Nosotros les vamos a hablar acerca de los fundamentos para los requerimientos del software,
00:08	en específico la definición de un requerimiento de software.
00:12	Para empezar, ¿qué es un requerimiento de software?
00:15	Un requerimiento de software es una descripción de lo que un sistema debe ser o cómo debe comportarse.
00:20	Por ejemplo, si queremos crear una aplicación de pedidos de comida,
00:24	un requerimiento funcional sería que el usuario debe poder seleccionar productos y realizar un pedido.
00:29	Y un requerimiento no funcional sería la aplicación debe cargar en menos de 3 segundos.
00:34	Esto nos ayuda a entender claramente qué se espera del sistema.
00:39	A continuación, otro ejemplo acerca de requerimientos funcionales.
00:43	Para entenderlo mejor, tenemos este ejemplo acerca de requerimientos funcionales aplicados a una aplicación móvil de banco,
00:49	donde tenemos tres puntos.
00:50	Primero, ¿qué hace el sistema?
00:51	Segundo, ¿cómo reacciona el sistema?
00:53	Y tercero, ¿qué no debe de hacer el sistema?
00:54	Para el primero, tenemos que el sistema debe permitir al usuario iniciar sesión con usuario y contraseña.
01:00	También que el sistema debe mostrar el saldo disponible después de iniciar sesión.
01:05	Y que el sistema debe permitir realizar transferencias a otras cuentas.
01:09	Para el segundo, tenemos que si el usuario ingresa la contraseña correctamente,
01:13	el sistema muestra el menú principal.
01:16	También tenemos que si el usuario ingresa una contraseña incorrecta,
01:18	el sistema muestra un mensaje de error.
01:21	Y si el usuario intenta transferir una cantidad no disponible,
01:26	el sistema muestra un mensaje de error y no realiza la operación.
01:30	Para el tercero, tenemos que el sistema no debe permitir el acceso con la contraseña incorrecta.
01:36	Otro sería que el sistema no debe permitir realizar transferencias a cuentas inexistentes.
01:41	Y por último, que el sistema no debe mostrar información de otras cuentas a usuarios no autorizados.
01:48	Este ejemplo muestra cómo los requerimientos funcionales describen qué servicios debe ofrecer al cliente,

01:55	cómo debe reaccionar ante entradas o situaciones, lo que no debe de hacer.
02:01	Todo esto asegura que el sistema cumpla con las necesidades del cliente de forma clara, completa y consistente.
02:07	El origen de los requerimientos surge de las necesidades del usuario o del problema que se quiere resolver.
02:14	Por eso es importante escuchar bien al cliente y entender qué necesita realmente, no sólo lo que cree que necesita.
02:21	Las características con las que cuenta un buen requerimiento es que debe ser claro, específico y fácil de entender.
02:29	También debe poder comprobarse, es decir, que se pueda verificar si se cumple o no.
02:34	Pasemos al ciclo básico de los requerimientos.
02:38	Primero se obtienen los requerimientos, por ejemplo, mediante entrevistas.
02:42	Después se analizan para ver si son viables.
02:45	Luego se documentan de forma clara.
02:47	Y finalmente se validan asegurando que realmente representen lo que el usuario necesita.
02:53	Es importante mencionar que los requerimientos tienen una gran importancia en nuestro proyecto,
02:59	ya que los requerimientos son la base de este mismo.
03:02	Si están mal definidos, el sistema puede terminar funcionando mal o no cumplir con lo que el usuario quería.
03:08	En cambio, si están bien hechos, facilitan todo el desarrollo.
03:11	Algunos errores comunes son hacer requerimientos ambiguos, no hablar con el usuario o no documentarlos correctamente.
03:20	Por ejemplo, decir el sistema debe ser rápido no es claro, pero decir debe responder en menos de dos segundos sí lo es.
03:30	Dentro de nuestras buenas prácticas, obviamente para evitar errores, es importante escribir requerimientos claros,
03:37	validar con el usuario y mantener una buena comunicación durante todo el proyecto.
03:44	En conclusión, los requerimientos de software son esenciales porque definen lo que se va a construir
03:49	y aseguran que el sistema cumpla con su objetivo.
03:54	Esperamos que este tutorial les haya servido para tener una idea más clara acerca de los fundamentos para los requerimientos del software,
04:01	específicamente definición de un requerimiento de software.
04:04	Hasta pronto.
04:07	Gracias.

6. Software interactivo

6.1. Objetivo

El presente tutorial tiene como objetivo sentar las bases para lo que son y lo que significan los requerimientos de software, en específico, para la ingeniería de software.

De tal forma que el lector podrá, al terminal el tutorial, gozar del entendimiento completo sobre lo que son los requerimientos de software, los tipos que existen, de qué dependen y cómo redactarlos, de manera teórica y práctica con los ejemplos mencionados. El lector podrá evaluar su entendimiento por medio del cuestionario presentado.

Al concluir el tutorial, el lector deberá ser capaz de:

- Comprender qué es un requerimiento funcional, distinguirlo de un requerimiento no funcional e identificar los factores contextuales —tipo de software, perfil del usuario y enfoque organizacional— que condicionan su redacción.
- Distinguir los niveles de granularidad (usuario y sistema) y las taxonomías más utilizadas para clasificar los requerimientos a lo largo del ciclo de vida del software.
- Aplicar plantillas y reglas de estilo para escribir requerimientos funcionales claros, así como valorar su calidad mediante criterios como verificabilidad, no ambigüedad y testabilidad.
- Etiquetar requerimientos con identificadore únicos y reconocer su red de relaciones con reglas de negocio, casos de uso, diseño y código, entendiendo el papel de la trazabilidad.

6.2. Banco de preguntas rápidas

#	Mal definido	Bien definido
1	El sistema debe procesar pedidos rápidamente	El sistema debe procesar pedidos en menos de 1.5 segundos
2	La aplicación debe ser intuitiva	El usuario debe poder completar una compra en máximo 3 pasos
3	El sistema debe manejar grandes volúmenes de datos	El sistema debe soportar hasta 10,000 registros simultáneos
4	La app debe funcionar bien	La app debe operar sin errores críticos durante 8 horas continuas
5	El sistema debe ser seguro	El sistema debe bloquear la cuenta tras 5 intentos fallidos
6	La página debe cargar rápido	La página debe cargar en menos de 2.5 segundos con conexión estándar
7	El sistema debe ser confiable	El sistema debe tener disponibilidad del 99 % mensual
8	La app debe ser flexible	El sistema debe permitir editar el perfil del usuario en cualquier momento
9	El sistema debe ser fácil de mantener	El sistema debe permitir actualizar módulos sin afectar otros componentes
10	El sistema debe ser moderno y atractivo	La interfaz debe seguir la guía de estilo Material Design
11	El sistema debe gestionar usuarios correctamente	El sistema debe permitir registrar, editar y eliminar usuarios

#	Mal definido	Bien definido
12	El sistema debe tener buen rendimiento	El sistema debe responder consultas en menos de 1 segundo
13	La aplicación debe ser amigable	La aplicación debe mostrar mensajes de error claros al usuario
14	El sistema debe ser escalable	El sistema debe permitir añadir nuevos módulos sin modificar el núcleo
15	El sistema debe manejar errores	El sistema debe registrar errores en un log con fecha y hora
16	La app debe ser rápida y eficiente	El sistema debe procesar 100 solicitudes por segundo
17	El sistema debe ser fácil de entender	El sistema debe incluir documentación accesible desde el menú principal
18	El sistema debe funcionar correctamente en todo momento	El sistema debe recuperarse automáticamente tras una falla en menos de 10 segundos
19	La aplicación debe ser compatible con dispositivos	La aplicación debe funcionar en Android 10 o superior
20	El sistema debe ser rápido al guardar datos	El sistema debe guardar datos en menos de 500 ms
21	El sistema debe tener buena seguridad	El sistema debe cifrar contraseñas usando SHA-256
22	La app debe ser estable	La app no debe cerrarse inesperadamente durante su uso
23	El sistema debe permitir múltiples usuarios	El sistema debe permitir hasta 50 usuarios concurrentes
24	El sistema debe ser rápido al iniciar	El sistema debe iniciar en menos de 3 segundos
25	El sistema debe ser adaptable	La interfaz debe ajustarse automáticamente a pantallas móviles
26	El sistema debe ser usable	El usuario debe poder completar un registro en menos de 2 minutos
27	El sistema debe ser eficiente en memoria	El sistema no debe consumir más de 200 MB de RAM
28	El sistema debe ser accesible	El sistema debe permitir navegación mediante teclado
29	El sistema debe ser confiable siempre	El sistema debe tener un tiempo medio entre fallos de 100 horas
30	El sistema debe ser fácil de instalar	El sistema debe instalarse en menos de 5 minutos
31	El sistema debe ser rápido al buscar	El sistema debe devolver resultados de búsqueda en menos de 1 segundo
32	El sistema debe ser claro	El sistema debe mostrar mensajes de confirmación al completar acciones

#	Mal definido	Bien definido
33	El sistema debe funcionar bien con internet lento	El sistema debe operar con conexiones de 1 Mbps
34	El sistema debe ser robusto	El sistema debe continuar funcionando ante entradas inválidas
35	El sistema debe ser organizado	El sistema debe clasificar los registros por fecha automáticamente
36	El sistema debe ser flexible para cambios	El sistema debe permitir modificar configuraciones sin reiniciar
37	El sistema debe ser amigable para todos	El sistema debe incluir opción de idioma español e inglés
38	El sistema debe funcionar en varios navegadores	El sistema debe funcionar en Chrome, Firefox y Edge
39	El sistema debe validar correctamente los datos	El sistema debe validar que el correo tenga formato válido antes de guardarlo
40	La aplicación debe ser rápida al mostrar resultados	La aplicación debe mostrar resultados en menos de 1 segundo tras la consulta
41	El sistema debe ser seguro contra ataques	El sistema debe bloquear direcciones IP tras 10 intentos fallidos en 1 minuto
42	El sistema debe tener buena interfaz	El sistema debe mostrar un menú principal con acceso a todas las funciones
43	El sistema debe ser fácil de navegar	El usuario debe poder acceder a cualquier sección en máximo 3 clics
44	El sistema debe manejar bien los errores	El sistema debe mostrar mensajes de error específicos para cada fallo
45	El sistema debe ser compatible con diferentes resoluciones	El sistema debe adaptarse a resoluciones desde 1280×720 hasta 1920×1080
46	El sistema debe ser confiable en el tiempo	El sistema debe mantener integridad de datos después de reinicios inesperados
47	El sistema debe ser eficiente al almacenar datos	El sistema debe comprimir archivos mayores a 5 MB antes de almacenarlos
48	El sistema debe ser accesible para todos	El sistema debe permitir aumentar el tamaño de fuente desde la configuración
49	El sistema debe responder correctamente a solicitudes	El sistema debe devolver código HTTP 200 en solicitudes exitosas
50	El sistema debe ser fácil de usar para principiantes	El sistema debe incluir un tutorial inicial para nuevos usuarios

7. Cuestionario

1. **Según los autores, ¿por qué los requerimientos son la base de todo proyecto de software?**

a) **Porque describen las funcionalidades y características que el software debe tener para satisfacer las necesidades de los usuarios y/o clientes (Celi-Párraga et al., 2023, p. 3).** *(Correct answer)*

b) Porque definen las pruebas unitarias que el equipo de calidad ejecutará durante la fase de verificación.

c) Porque establecen el lenguaje de programación y la arquitectura de despliegue del producto final.

d) Porque determinan el presupuesto y el cronograma de entrega comprometido con el cliente.

2. **¿Cuál es la advertencia que hacen Contreras & Hernández respecto a la dicotomía funcional/no funcional?**

a) Que los requerimientos no funcionales deben redactarse siempre antes que los funcionales.

b) **Que con frecuencia se mencionan dos grandes tipos de requerimientos, funcionales y no funcionales, aunque conviene tener cuidado con estas definiciones, puesto que no es raro encontrar que exista una relación entre ambas más cercana de lo que se espera (Contreras Bernal & Hernández Ruiz, 2022, p. 93).** *(Correct answer)*

c) Que la separación entre funcionales y no funcionales es absoluta y nunca debe relativizarse.

d) Que los requerimientos no funcionales son siempre opcionales mientras que los funcionales son obligatorios.

3. **¿Cuál es la frase central de Sommerville sobre los requerimientos funcionales que casi todos los autores latinoamericanos recogen?**

a) Los requerimientos funcionales son atributos de calidad del sistema que deben ser medibles cuantitativamente.

b) **Los requerimientos funcionales para un sistema refieren lo que el sistema debe hacer (Sommerville, 2011, §4.1.1, p. 85).** *(Correct answer)*

c) Los requerimientos funcionales describen exclusivamente las restricciones operacionales del sistema.

d) Los requerimientos funcionales son las condiciones contractuales entre el cliente y el desarrollador.

4. **¿Cómo formula Sommerville de manera extendida los requerimientos funcionales?**

a) Como diagramas UML que representan la estructura interna del sistema y sus dependencias.

b) Como listas de chequeo derivadas de los estándares ISO 9001 aplicados al desarrollo de software.

c) **Como 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 (Sommerville, 2011, §4.1, p. 84). *(Correct answer)*

d) Como métricas de rendimiento que miden la latencia y el throughput del sistema bajo carga.

5. **Según Celi-Párraga et al., ¿cómo se definen los requerimientos funcionales?**

   a) Como restricciones impuestas por el hardware sobre la lógica de negocio del sistema.

   b) Como criterios de aceptación firmados por el patrocinador del proyecto.

   c) Como contratos legales entre el equipo de desarrollo y el área de operaciones.

   d) **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 (Celi-Párraga et al., 2023, §3.3, p. 39).** *(Correct answer)*

6. **¿Qué frase-resumen ofrecen Celi-Párraga et al. tomada directamente de Sommerville?**

   a) **Los requerimientos funcionales para un sistema refieren lo que el sistema debe hacer (Celi-Párraga et al., 2023, §3.3, p. 39).** *(Correct answer)*

   b) Los requerimientos funcionales son los componentes mínimos verificables del sistema.

   c) Los requerimientos funcionales son la traducción técnica de los objetivos del negocio.

   d) Los requerimientos funcionales son siempre derivados de los requerimientos no funcionales.

7. **¿Cómo describe Laplante los *functional requirements* (FRs) de manera industrial pero compacta?**

   a) Como las propiedades de calidad medibles del producto en producción.

   b) Como los algoritmos específicos que el equipo de desarrollo implementará.

   c) **Los *functional requirements* (FRs) describen los servicios que el sistema debe proveer y cómo el sistema reaccionará ante sus entradas (Laplante, 2017, Cap. 1, p. 6).** *(Correct answer)*

   d) Como las interfaces gráficas con las que el usuario interactuará.

8. **¿Qué énfasis añade Laplante, igual que Sommerville, sobre lo que el sistema no debe hacer?**

   a) **Los requerimientos funcionales necesitan establecer explícitamente ciertos comportamientos que el sistema no debe ejecutar (Laplante, 2017, Cap. 1, p. 6).** *(Correct answer)*

   b) Los requerimientos funcionales solo deben describir comportamientos positivos del sistema.

   c) Los requerimientos funcionales nunca deben mencionar restricciones negativas porque generan ambigüedad.

d) Los requerimientos funcionales deben enumerar exclusivamente los casos de éxito del sistema.

9. **Según SWEBOK, ¿qué describen los requerimientos funcionales?**

   a) Las propiedades de mantenibilidad y portabilidad que el software debe exhibir.

   b) **Los requerimientos funcionales describen las funciones que el software debe ejecutar; por ejemplo, formatear un texto o modular una señal; a veces se les conoce como *capabilities* o *features* (Bourque & Fairley, 2014, Software Requirements KA, §1.3, p. 1-3).** *(Correct answer)*

   c) Los protocolos de comunicación entre los componentes distribuidos del sistema.

   d) La estructura de la base de datos relacional y sus restricciones de integridad.

10. **¿Cuál es el criterio de identificación operativo que aporta SWEBOK para los requerimientos funcionales?**

    a) Que sean comprensibles únicamente por el equipo de arquitectura del sistema.

    b) Que se redacten exclusivamente con verbos en voz pasiva para evitar ambigüedad.

    c) Que estén numerados según el estándar ANSI/IEEE 830 sin excepciones.

    d) **Un requerimiento funcional puede describirse como aquél para el cual puede escribirse un conjunto finito de pasos de prueba que validen su comportamiento (Bourque & Fairley, 2014, Software Requirements KA, §1.3, p. 1-3).** *(Correct answer)*

11. **Según SWEBOK, ¿cuál es la propiedad esencial de todos los requerimientos de software?**

    a) Que estén traducidos al idioma del cliente final como mínimo en dos versiones.

    b) Que sean escritos por un ingeniero certificado en metodologías ágiles.

    c) **Una propiedad esencial de todos los requerimientos de software es que sean verificables como característica individual en el caso de un requerimiento funcional, o a nivel del sistema en el caso de un requerimiento no funcional (Bourque & Fairley, 2014, Software Requirements KA, §1.1, p. 1-2).** *(Correct answer)*

    d) Que tengan asociado un costo estimado en horas-persona desde su redacción inicial.

12. **¿Cuál es la formulación más breve y precisa de un requerimiento funcional según Wiegers & Beatty?**

    a) **Un requerimiento funcional es una descripción de un comportamiento que un sistema exhibirá bajo condiciones específicas (Wiegers & Beatty, 2013, Tabla 1-1, p. 7).** *(Correct answer)*

    b) Un requerimiento funcional es una promesa contractual de

    El presente tutorial tiene como objetivo sentar las bases para lo que son y lo que siginifcan los requerimientos funcionales para el desarrollo de software, en específico, para la ingeneiría de software.

De tal forma que el lector podrá, al terminar el tutorial, gozar del entenedimiento completo sobre lo que son los requerimientos de software, los tipos que existen, de qué depende y cómo redactarlos.

, de manera teórica y práctica con los ejemplos mencionados.

El lector podrá evaluar su entendimiento por medio del cuestionario presentado.l proveedor frente al cliente.

c) Un requerimiento funcional es la implementación detallada de un caso de uso del sistema.

d) Un requerimiento funcional es una métrica cuantitativa del rendimiento del sistema.

13. **¿Qué precisa la versión ampliada de la definición de Wiegers sobre los tres niveles de requerimientos?**

a) Que los requerimientos funcionales deben escribirse después del código fuente.

b) Que los requerimientos funcionales se redactan únicamente por el equipo de pruebas.

c) **Los requerimientos funcionales especifican los comportamientos que el producto exhibirá bajo condiciones específicas, describiendo lo que los desarrolladores deben implementar para que los usuarios puedan cumplir sus tareas (requerimientos del usuario), satisfaciendo así los requerimientos del negocio; esta alineación entre los tres niveles de requerimientos es esencial para el éxito del proyecto (Wiegers & Beatty, 2013, p. 9).** *(Correct answer)*

d) Que los requerimientos funcionales son irrelevantes para los requerimientos del negocio.

14. **Según Robertson & Robertson, ¿qué es en esencia un requerimiento funcional?**

a) Una restricción técnica derivada de la plataforma elegida para el despliegue.

b) Un atributo de calidad medible bajo condiciones de prueba controladas.

c) Un fragmento de código documentado que implementa una función del sistema.

d) **En esencia, un requerimiento funcional es algo que el producto debe hacer para soportar el negocio de su propietario (Robertson & Robertson, 2012, Cap. 1, Truth 10, p. 8).** *(Correct answer)*

15. **¿Cuál es la definición canónica de apertura del Capítulo 10 de Robertson & Robertson?**

a) **Los requerimientos funcionales especifican lo que el producto debe hacer: las acciones que debe realizar para satisfacer las razones fundamentales de su existencia (Robertson & Robertson, 2012, Cap. 10, p. 223).** *(Correct answer)*

b) Los requerimientos funcionales son los atributos no negociables del contrato comercial.

c) Los requerimientos funcionales describen únicamente la interfaz gráfica del usuario final.

d) Los requerimientos funcionales son las restricciones legales aplicables al producto.

16. **Según Contreras & Hernández, ¿cómo se definen los requerimientos funcionales?**

    a) Como las pruebas de aceptación firmadas por el cliente al final del proyecto.

    b) Como los componentes reutilizables del catálogo organizacional de patrones.

    c) **Aquellos que describen con detalle lo que el sistema debe hacer, es decir, su comportamiento a partir de indicaciones sobre cuáles son las excepciones, entradas y salidas del mismo; por lo anterior, dependen mucho del tipo de software, de los usuarios, así como de lo específico y particular que pueda ser el negocio de la organización (Contreras Bernal & Hernández Ruiz, 2022, p. 93).** *(Correct answer)*

    d) Como las cláusulas penales del contrato de desarrollo de software.

17. **¿Cuál es el esquema de identificación que propone Pressman para clasificar requisitos?**

    a) A = aceptación, B = base de datos, C = código, D = despliegue, E = entrega.

    b) P = prueba, Q = calidad, R = rendimiento, S = seguridad, T = tiempo.

    c) U = usuario, V = vista, W = workflow, X = excepción, Y = yield.

    d) **F = requisito funcional, D = requisito de datos, C = requisito de comportamiento, Z = requisito de interfaz, y S = requisito de salida; un requisito identificado como F09 indica que se trata de un requisito funcional y que tiene asignado el número 9 dentro de los citados requisitos (Pressman, 2002, §10.5.6, p. 175).** *(Correct answer)*

18. **¿Cómo define Pressman la calidad del software por contraste?**

    a) Como la satisfacción del cliente medida en encuestas posteriores al lanzamiento del producto.

    b) **La calidad del software se define como concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados, y con las características implícitas que se espera de todo software desarrollado profesionalmente (Pressman, 2002, Cap. 8, p. 135).** *(Correct answer)*

    c) Como el cumplimiento exclusivo del cronograma de entrega original del proyecto.

    d) Como la cantidad de líneas de código documentadas con comentarios en línea.

19. **Según Sommerville, ¿de qué dependen los requerimientos funcionales?**

    a) Del estilo personal de redacción del analista de requerimientos asignado al proyecto.

    b) **Los requerimientos funcionales dependen del tipo de software que se esté desarrollando, de los usuarios esperados del software y del enfoque general que adopta la organización cuando se escriben los requerimientos (Sommerville, 2011, §4.1.1, p. 85).** *(Correct answer)*

    c) Exclusivamente de la plataforma tecnológica seleccionada para el despliegue.

    d) Únicamente del presupuesto disponible para el proyecto en el ciclo fiscal.

20. **¿Cuál es la formulación de Sommerville sobre el modelo binario clásico usuario/sistema?**

    a) Que los requerimientos del usuario son siempre más detallados que los del sistema.

    b) Que los requerimientos funcionales solo aplican al nivel de sistema, nunca al de usuario.

    c) **Al expresarse como requerimientos del usuario, los requerimientos funcionales se describen por lo general de forma abstracta que entiendan los usuarios del sistema; sin embargo, requerimientos funcionales más específicos del sistema detallan las funciones del sistema, sus entradas y salidas, sus excepciones, etcétera (Sommerville, 2011, §4.1.1, p. 85).** *(Correct answer)*

    d) Que el modelo binario fue descartado por la literatura moderna en favor de uno ternario.

21. **¿Cómo formulan Celi-Párraga la diferencia entre requerimientos del usuario y del sistema?**

    a) **Los requerimientos del usuario son enunciados, en un lenguaje natural junto con diagramas, acerca de qué servicios esperan los usuarios del sistema, y de las restricciones con las cuales éste debe operar; los requerimientos del sistema son descripciones más detalladas de las funciones, los servicios y las restricciones operacionales del sistema de software (Celi-Párraga et al., 2023, §3.2, pp. 37–38).** *(Correct answer)*

    b) Los requerimientos del usuario son atributos de calidad mientras que los del sistema son funcionales.

    c) Los requerimientos del usuario son verbales mientras que los del sistema son siempre matemáticos.

    d) Los requerimientos del usuario y del sistema son sinónimos intercambiables sin diferencia técnica.

22. **Según Sommerville, ¿qué rango interno tienen los requerimientos funcionales del sistema?**

    a) **Los requerimientos funcionales del sistema varían desde requerimientos generales que cubren lo que tiene que hacer el sistema, hasta requerimientos muy específicos que reflejan maneras locales de trabajar o los sistemas existentes de una organización (Sommerville, 2011, §4.1.1, p. 85).** *(Correct answer)*

    b) Los requerimientos funcionales del sistema son uniformes y siempre del mismo nivel de detalle.

    c) Los requerimientos funcionales del sistema son siempre más generales que los del usuario.

    d) Los requerimientos funcionales del sistema solo aplican a sistemas embebidos críticos.

23. **¿Cómo reproduce Laplante la estratificación clásica usuario/sistema?**

a) Indicando que los requerimientos funcionales son siempre del mismo nivel sin estratificación posible.

b) Indicando que la estratificación clásica solo aplica a sistemas distribuidos de gran escala.

c) **Los requerimientos funcionales pueden ser de alto nivel y generales —en cuyo caso son requerimientos del usuario— o pueden ser detallados, expresando entradas, salidas, excepciones, etcétera —en cuyo caso son requerimientos del sistema (Laplante, 2017, Cap. 1, p. 6).** *(Correct answer)*

d) Indicando que los requerimientos detallados son siempre del usuario y los generales del sistema.

24. **¿Qué tres niveles distintos de requerimientos del software introducen Wiegers & Beatty?**

a) Niveles de prioridad alta, media y baja según el impacto en el negocio.

b) Niveles de redacción técnica, semitécnica y no técnica según la audiencia.

c) **Los requerimientos del software incluyen tres niveles distintos: requerimientos de negocio, requerimientos del usuario y requerimientos funcionales; adicionalmente, todo sistema tiene una colección de requerimientos no funcionales (Wiegers & Beatty, 2013, p. 7).** *(Correct answer)*

d) Niveles de validación, verificación y aceptación a lo largo del ciclo de vida.

25. **Según Wiegers & Beatty, ¿qué es una *feature*?**

a) **Una *feature* consiste en una o más capacidades del sistema lógicamente relacionadas que proveen valor a un usuario y son descritas por un conjunto de requerimientos funcionales (Wiegers & Beatty, 2013, p. 7).** *(Correct answer)*

b) Una *feature* es un fragmento de código en un repositorio Git aislado.

c) Una *feature* es un atributo no funcional medido en términos de tiempo de respuesta.

d) Una *feature* es un documento legal que firma el cliente al recibir el producto.

26. **¿Cómo describen Wiegers & Beatty la distinción de perspectiva entre requerimientos del usuario y funcionales?**

a) Los requerimientos del usuario son técnicos y los funcionales son comerciales.

b) **Los requerimientos del usuario describen la vista del usuario; los requerimientos funcionales describen la vista del desarrollador (Wiegers & Beatty, 2013, p. 89).** *(Correct answer)*

c) Los requerimientos del usuario son obligatorios y los funcionales son opcionales.

d) Los requerimientos del usuario y los funcionales coinciden completamente en su perspectiva.

27. **¿Qué jerarquía no subjetiva proponen Robertson & Robertson?**

a) Una jerarquía de tres niveles: estratégico, táctico y operacional con asignación de roles.

b) Una jerarquía de cuatro niveles basada en el modelo de madurez CMMI clásico.

c) **Una jerarquía no subjetiva de cinco niveles —contexto de trabajo, evento de negocio, caso de uso de producto, paso del caso de uso de producto, y requerimiento atómico—. Los requerimientos funcionales bien redactados son requerimientos atómicos: ayuda a pensar en los requerimientos atómicos como el nivel más bajo de especificación de requerimientos (Robertson & Robertson, 2012, Cap. 10, p. 238).** *(Correct answer)*

d) Una jerarquía plana sin subordinación entre tipos de requerimientos.

28. **¿Cómo describen Celi-Párraga los requerimientos no funcionales?**

a) **Los requerimientos no funcionales son limitaciones sobre servicios o funciones que ofrece el sistema; incluyen restricciones tanto de temporización y del proceso de desarrollo, como impuestas por los estándares; los requerimientos no funcionales se suelen aplicar al sistema como un todo, más que a características o a servicios individuales del sistema (Celi-Párraga et al., 2023, §3.3, p. 40).** *(Correct answer)*

b) Los requerimientos no funcionales son siempre opcionales y pueden omitirse del SRS.

c) Los requerimientos no funcionales describen únicamente la interfaz visual del producto.

d) Los requerimientos no funcionales son siempre derivables a partir del código fuente final.

29. **Según Celi-Párraga, ¿qué dice el complemento sobre los requerimientos no funcionales?**

a) Que los requerimientos no funcionales coinciden completamente con los servicios al usuario.

b) **Los requerimientos no funcionales, como indica su nombre, son requerimientos que no se relacionan directamente con los servicios específicos que el sistema entrega a sus usuarios (Celi-Párraga et al., 2023, §3.3, p. 41).** *(Correct answer)*

c) Que los requerimientos no funcionales solo aplican a sistemas en la nube.

d) Que los requerimientos no funcionales son sustitutos directos de los funcionales.

30. **¿En qué tres grandes categorías se subdividen los NFR según Celi-Párraga?**

a) Categorías de prioridad alta, prioridad media y prioridad baja según importancia.

b) **Los NFR se subdividen en tres grandes categorías: del producto (usabilidad, eficiencia, dependibilidad, seguridad), organizacionales (entorno, organizacionales, desarrollo) y externos (regulatorios, éticos, legislativos) (Celi-Párraga et al., 2023, §3.3, p. 41).** *(Correct answer)*

c) Categorías cuantitativas, cualitativas y mixtas según su forma de medición técnica.

d) Categorías obligatorias, recomendables y opcionales según el contrato firmado.

31. **¿Qué advertencia da Sommerville sobre la frontera difusa entre FR y NFR?**

a) Que los NFR siempre deben expresarse junto con los FR y nunca por separado.

b) **Si los requerimientos no funcionales se expresan por separado de los requerimientos funcionales, las relaciones entre ambos serían difíciles de entender (Sommerville, 2011, Cap. 4, p. 90).** *(Correct answer)*

c) Que los NFR y FR son completamente independientes y no requieren ser correlacionados.

d) Que los NFR siempre deben omitirse cuando los FR ya están bien documentados.

32. **Según SWEBOK, ¿cómo se relacionan estructuralmente los requerimientos de calidad con los funcionales?**

a) Los requerimientos de calidad sustituyen completamente a los requerimientos funcionales.

b) Los requerimientos de calidad son independientes de los requerimientos funcionales.

c) **Los requerimientos de calidad del software son en realidad atributos de (o restricciones sobre) los requerimientos funcionales (lo que el sistema hace); este KA busca claridad usando software quality en el sentido más amplio y usando los requerimientos de calidad del software como restricciones sobre los requerimientos funcionales (Bourque & Fairley, 2014, Software Quality KA, p. 10-1).** *(Correct answer)*

d) Los requerimientos de calidad solo aplican al hardware del sistema, no al software.

33. **¿Cómo caracteriza SWEBOK los requerimientos no funcionales?**

a) **Los requerimientos no funcionales son aquellos que actúan para restringir la solución; a veces se les conoce como *constraints* o *quality requirements*; pueden clasificarse adicionalmente en requerimientos de rendimiento, mantenibilidad, seguridad (safety), confiabilidad, seguridad (security), interoperabilidad u otros (Bourque & Fairley, 2014, Software Requirements KA, §1.3, p. 1-3).** *(Correct answer)*

b) Los requerimientos no funcionales son sinónimos exactos de los requerimientos del usuario.

c) Los requerimientos no funcionales solo se refieren a aspectos legales del sistema.

d) Los requerimientos no funcionales se clasifican exclusivamente en dos categorías: técnicos y comerciales.

34. **¿Cómo se asocian los NFR con los requerimientos funcionales según Laplante?**

a) **Los NFR pueden asociarse a los requerimientos funcionales, por ejemplo: «solo usuarios autorizados deberían poder acceder a la funcionalidad X del sistema» (Laplante, 2017, p. 10).** *(Correct answer)*

b) Los NFR nunca pueden asociarse a los requerimientos funcionales por su naturaleza.

c) Los NFR siempre se asocian uno a uno con cada requerimiento funcional sin excepciones.

d) Los NFR son completamente equivalentes a los requerimientos funcionales sin distinción.

35. **¿Qué ejemplo paradigmático dan Wiegers & Beatty sobre la trazabilidad NFR→FR?**

a) **Los requerimientos de seguridad para autenticación de usuarios dan lugar a requerimientos funcionales derivados que podrían implementarse a través de funcionalidad de contraseñas o biometría (Wiegers & Beatty, 2013, p. 290).** *(Correct answer)*

b) Los requerimientos de rendimiento siempre se traducen literalmente en código sin pasar por FR.

c) Los requerimientos de usabilidad nunca generan requerimientos funcionales derivados.

d) Los requerimientos de mantenibilidad son sustituidos directamente por los funcionales.

36. **¿Cuál es la taxonomía de tres tipos que propone Laplante?**

a) Requerimientos técnicos, comerciales y legales según el ámbito de aplicación.

b) Requerimientos del cliente, del proveedor y del usuario final según los actores.

c) **Otra taxonomía para las especificaciones de requerimientos se enfoca en el tipo de requerimiento de la siguiente lista de posibilidades: requerimientos funcionales (FRs), requerimientos no funcionales (NFRs) y requerimientos de dominio (Laplante, 2017, Cap. 1, p. 6).** *(Correct answer)*

d) Requerimientos obligatorios, deseables y opcionales según la prioridad asignada.

37. **¿Cómo describe Laplante los requerimientos de dominio?**

a) **Los requerimientos de dominio son derivados del dominio de aplicación; estos tipos de requerimientos pueden consistir en nuevos requerimientos funcionales o restricciones a requerimientos funcionales existentes, o pueden especificar cómo deben realizarse cómputos particulares (Laplante, 2017, p. 11).** *(Correct answer)*

b) Los requerimientos de dominio son requerimientos derivados exclusivamente del marketing del producto.

c) Los requerimientos de dominio son una invención hispanoamericana sin equivalente internacional.

d) Los requerimientos de dominio son sinónimos exactos de los requerimientos no funcionales.

38. **¿Cómo formulan Celi-Párraga el principio rector sobre la redacción de requerimientos del usuario?**

a) Que los requerimientos del usuario deben escribirse en notación matemática formal exclusivamente.

b) Que los requerimientos del usuario deben incluir detalles de la arquitectura del sistema.

c) Que los requerimientos del usuario deben usar lenguaje técnico especializado para garantizar precisión.

d) **Los requerimientos del usuario para un sistema deben describir los requerimientos funcionales y no funcionales, de forma que sean comprensibles para los usuarios del sistema que no cuentan con un conocimiento técnico detallado; el documento de requerimientos no debe incluir detalles de la arquitectura o el diseño del sistema; en consecuencia, si se escriben los requerimientos del usuario, no se debe usar jerga de software, anotaciones estructuradas o formales; deben escribirse los requerimientos del usuario en lenguaje natural, con tablas y formas sencillas, así como diagramas intuitivos (Celi-Párraga et al., 2023, §3.5, p. 44).** *(Correct answer)*

39. **Según Celi-Párraga, ¿cómo cambian las herramientas a nivel del sistema?**

a) **Los requerimientos del usuario se escriben casi siempre en lenguaje natural, complementado con diagramas y tablas adecuados en el documento de requerimientos; los requerimientos del sistema se escriben también en lenguaje natural, pero de igual modo se utilizan otras notaciones basadas en formas, modelos gráficos del sistema o modelos matemáticos del sistema (Celi-Párraga et al., 2023, §3.5, p. 44).** *(Correct answer)*

b) Los requerimientos del sistema deben escribirse exclusivamente en pseudocódigo verificable.

c) Los requerimientos del sistema solo pueden escribirse en inglés técnico estandarizado.

d) Los requerimientos del sistema y del usuario deben escribirse en formatos completamente disjuntos.

40. **¿Cuáles son los cinco lineamientos que recomienda Sommerville para minimizar la interpretación errónea?**

a) (i) uso obligatorio de pseudocódigo; (ii) todos los verbos en pasado; (iii) sin negaciones; (iv) máximo cinco palabras por requerimiento; (v) sin numeración.

b) **(i) formato estándar uniforme para todos los requerimientos —un requerimiento, una oración— asociado con su razón y fuente; (ii) distinguir obligatorio de deseable mediante verbos consistentes («debe» vs. «debería»); (iii) resaltar las partes clave con negrilla, cursiva o color; (iv) evitar la jerga técnica; (v) asociar una razón a cada requerimiento de usuario para facilitar el cambio futuro (Sommerville, 2011, p. 96).** *(Correct answer)*

c) (i) exclusivamente lenguaje natural; (ii) sin formato visual; (iii) con jerga técnica detallada; (iv) sin razones; (v) sin estructura uniforme.

d) (i) siempre en idioma inglés; (ii) con notación UML obligatoria; (iii) sin distinguir prioridad; (iv) con verbos imperativos; (v) con anexos legales obligatorios.

41. **¿Qué información debe incluir la plantilla estructurada de siete campos de Sommerville?**

a) (1) hora; (2) fecha; (3) lugar; (4) persona; (5) equipo; (6) tarea; (7) estado.

b) (1) costo; (2) plazo; (3) recursos; (4) prioridad; (5) riesgo; (6) impacto; (7) beneficio.

c) **(1) función o entidad —descripción de qué se especifica; (2) entradas y procedencias; (3) salidas y destinos; (4) requiere —datos u otras entidades del sistema necesarios para el cálculo; (5) acción —qué se va a hacer;**

(6) precondición y postcondición; (7) efectos colaterales (Sommerville, 2011, §4.3.2, p. 98). *(Correct answer)*

d) (1) título; (2) autor; (3) fecha; (4) versión; (5) estado; (6) aprobador; (7) revisor.

42. **Según Robertson & Robertson, ¿cuál es el nivel de detalle revelador en sus ejemplos?**

a) **Los requerimientos están escritos como una única oración con un único verbo; cuando se escribe esta única oración, se hace el requerimiento mucho más fácil de probar y mucho menos susceptible a la ambigüedad (Robertson & Robertson, 2012, Cap. 10, p. 228).** *(Correct answer)*

b) Los requerimientos están escritos como párrafos extensos con múltiples verbos coordinados.

c) Los requerimientos están escritos exclusivamente como diagramas UML sin texto narrativo.

d) Los requerimientos están escritos sin verbos para evitar ambigüedad sintáctica.

43. **¿Cuál es la forma canónica que recomiendan Robertson & Robertson?**

a) «Could be considered to...» para mantener flexibilidad en la implementación.

b) «It would be nice if...» para no presionar al equipo de desarrollo.

c) «The system might possibly...» para dejar abierta la prioridad del requerimiento.

d) **La forma canónica es «The product shall...»: hace la oración activa y se enfoca en comunicar lo que el producto pretende hacer (Robertson & Robertson, 2012, p. 228).** *(Correct answer)*

44. **¿Por qué desaconsejan Robertson & Robertson mezclar verbos modales como shall, must, will, might, could?**

a) Porque la lengua inglesa no admite combinaciones de modales en un mismo documento.

b) **A veces las personas usan una mezcla de «shall», «must», «will», «might», «could», etc., para indicar la prioridad de un requerimiento; esta práctica resulta en confusión semántica y se aconseja contra ella; en su lugar, debe usarse una forma consistente para escribir las descripciones de los requerimientos —«The product shall...» es la más común— y usar un atributo separado del requerimiento para identificar su prioridad (Robertson & Robertson, 2012, p. 229).** *(Correct answer)*

c) Porque los verbos modales están prohibidos por el estándar IEEE 830 vigente.

d) Porque los verbos modales solo aplican a requerimientos de calidad pero no a funcionales.

45. **¿Qué exigen Robertson & Robertson sobre el rationale en cada requerimiento?**

a) Que el rationale sustituya por completo a la descripción del requerimiento.

b) Que el rationale solo se incluya cuando el requerimiento sea de baja prioridad.

c) **Se sugiere —fuertemente— que se añada un *rationale* a los requerimientos para mostrar por qué existe el requerimiento; en algunos casos puede ser obvio, pero en muchas circunstancias es un componente crucial del requerimiento (Robertson & Robertson, 2012, p. 229).** *(Correct answer)*

d) Que el rationale debe redactarse en pseudocódigo formal exclusivamente.

46. **¿Cuál es el ejemplo concreto de Description + Rationale dado por Robertson & Robertson?**

a) **Description: «The product shall record roads that have been treated.» Rationale: «To be able to schedule untreated roads and highlight potential danger.» (Robertson & Robertson, 2012, pp. 229–230).** *(Correct answer)*

b) Description: «El sistema procesará pagos.» Rationale: «Porque el cliente lo necesita.»

c) Description: «The product manages users.» Rationale: «It is required by law.»

d) Description: «Sistema de inventario.» Rationale: «Para vender más productos.»

47. **Según Laplante (IEEE 29148), ¿qué deben capturar los requerimientos funcionales?**

a) **Los requerimientos funcionales deben capturar todas las entradas del sistema y la secuencia exacta de operaciones y respuestas (salidas) ante situaciones normales y anormales para cada posibilidad de entrada (Laplante, 2017, Cap. 4, p. 98).** *(Correct answer)*

b) Los requerimientos funcionales deben capturar exclusivamente los casos de éxito felices.

c) Los requerimientos funcionales deben capturar únicamente las restricciones legales aplicables.

d) Los requerimientos funcionales deben capturar solo los flujos de la interfaz gráfica.

48. **Según Laplante, ¿cuáles son las opciones organizacionales que provee IEEE 29148 para los requerimientos funcionales?**

a) **IEEE 29148 provee un número de opciones organizacionales para los requerimientos funcionales; los requerimientos funcionales pueden organizarse por: modo del sistema, clase de usuario, objeto, característica, estímulo, jerarquía funcional (Laplante, 2017, p. 99).** *(Correct answer)*

b) IEEE 29148 obliga a organizar los requerimientos funcionales únicamente por orden alfabético del autor.

c) IEEE 29148 prohíbe cualquier organización por estímulo o respuesta del sistema.

d) IEEE 29148 solo permite organización por costo estimado y prioridad de implementación.

49. **¿Qué convención más rica que la de Pressman ofrecen Wiegers & Beatty para identificar requerimientos?**

a) Identificación obligatoria mediante hash criptográfico SHA-256 generado del texto.

b) Identificación únicamente por nombre del autor seguido de la fecha de redacción.

c) **El enfoque más simple da a cada requerimiento un número de secuencia único, como UC-9 o FR-26; el prefijo indica el tipo de requerimiento, como FR para requerimiento funcional (Wiegers & Beatty, 2013, p. 187).** *(Correct answer)*

d) Identificación obligatoria mediante código de barras impreso en cada documento.

50. **¿Qué buena convención sugieren Wiegers & Beatty para los requerimientos jerárquicos?**

a) Que los requerimientos padre y los hijos sean idénticos para mantener consistencia.

b) **Una buena convención es escribir el requerimiento padre para que parezca un título, encabezado o nombre de feature, en lugar de parecerse a un requerimiento funcional en sí mismo; los requerimientos hijos de ese padre, en agregado, entregan la capacidad descrita en el padre (Wiegers & Beatty, 2013, p. 188).** *(Correct answer)*

c) Que los requerimientos jerárquicos no deben usarse en proyectos profesionales.

d) Que el requerimiento padre debe ser el más detallado y los hijos los más generales.

51. **¿Cuáles son los atributos individuales de un FR excelente según Wiegers & Beatty?**

a) Brevedad, oscuridad, ambigüedad, abstracción, generalidad, opcionalidad y rigidez.

b) **Completitud (*Complete*), corrección (*Correct*), factibilidad (*Feasible*), necesidad (*Necessary*), priorización (*Prioritized*), no ambigüedad (*Unambiguous*) y verificabilidad (*Verifiable*) (Wiegers & Beatty, 2013, Cap. 11, pp. 204–205).** *(Correct answer)*

c) Velocidad, escalabilidad, seguridad, usabilidad, accesibilidad, portabilidad y reusabilidad.

d) Económico, rápido, barato, simple, popular, atractivo y comercial.

52. **Según Wiegers & Beatty, ¿cuáles son los atributos del conjunto de FRs?**

a) Que los FRs deben ser idénticos entre sí para garantizar uniformidad sin excepción.

b) **Completitud (no faltan FRs), consistencia (no se contradicen entre sí), modificabilidad (se pueden modificar manteniendo histórico) y trazabilidad (bidireccional) (Wiegers & Beatty, 2013, p. 206).** *(Correct answer)*

c) Que los FRs deben ser inmutables y nunca modificarse después de redactados.

d) Que los FRs deben numerarse en orden estrictamente alfabético sin repetición.

8. Conclusión

Recorrida la literatura, conviene detenerse y mirar el camino con perspectiva. Lo primero que sorprende —y que el tutorial ha buscado mostrar sin disimular— es que el concepto de «requerimiento funcional», lejos de ser una idea cerrada y monolítica, es un campo de matices. Cada autor analizado ofrece una definición ligeramente distinta y, sin embargo, todas convergen en lo esencial: un requerimiento funcional describe lo que el sistema —o el producto, según la

tradición— debe hacer en respuesta a entradas y situaciones específicas, incluyendo, cuando aplica, lo que no debe hacer.

La aparente diversidad de definiciones no es un defecto del campo, sino un reflejo de la riqueza del concepto. Sommerville aporta la formulación clásica, breve y memorable. Celi-Párraga la traduce y adapta al contexto hispanoamericano sin pérdida de rigor. Laplante añade la precisión operativa de los estándares industriales y la disciplina de IEEE 29148. SWEBOK condensa el consenso normativo y eleva la testabilidad a criterio distintivo. Wiegers & Beatty introducen el lenguaje del comportamiento observable bajo condiciones específicas, junto con el modelo de tres niveles que distingue requerimientos de negocio, de usuario y funcionales. Robertson & Robertson ofrecen el aparato metodológico más estricto, exigiendo una sola oración con un solo verbo activo y un rationale explícito que justifique cada requerimiento. Pressman, por último, recuerda que un requerimiento funcional también es un objeto de gestión, de medición y de prueba dentro del ciclo de vida del software.

El tutorial ha tratado de mostrar que estas voces no compiten, sino que se complementan. Quien aprende a leerlas en conjunto obtiene algo más valioso que una sola definición: obtiene un repertorio de criterios que puede aplicar según el contexto del proyecto. Para un proyecto pequeño y ágil, basta con la disciplina mínima de la testabilidad y de la oración única. Para un proyecto grande, formal y crítico, se justifica el aparato completo —plantillas estructuradas, identificadores trazables, rationale explícito, atributos de calidad, jerarquías formales—. La sabiduría no está en aplicar siempre la versión más exigente, sino en saber cuál usar cuándo.

De toda la discusión emergen, a nuestro juicio, cinco ideas que el lector debería llevarse consigo. La primera es que un requerimiento funcional es, antes que cualquier otra cosa, un acuerdo: un acuerdo entre quien encarga el sistema y quien lo construye. Por eso debe escribirse en un lenguaje que ambos puedan entender, y por eso la ambigüedad es su peor enemigo. La segunda es que la testabilidad no es una propiedad opcional ni un lujo de proyectos formales: es la única forma objetiva de saber si un requerimiento está bien planteado. Si no se puede pensar una prueba que lo verifique, hay que reescribirlo. La tercera es que un buen requerimiento describe necesidades, no soluciones; el momento en que el analista escribe «el sistema usará una base de datos PostgreSQL» en lugar de «el sistema almacenará los registros de forma persistente y recuperable» es el momento en que ha dejado de hacer ingeniería de requerimientos para empezar a hacer diseño prematuro. La cuarta es que la frontera entre lo funcional y lo no funcional es porosa, y que ignorar las relaciones entre ambos tipos —especialmente cuando un requerimiento no funcional, como la seguridad, genera requerimientos funcionales derivados— es renunciar a entender el sistema como un todo. La quinta, finalmente, es que la trazabilidad, tantas veces percibida como un trámite burocrático, es en realidad lo que permite que un proyecto sobreviva al paso del tiempo y al recambio de personas: un requerimiento sin trazabilidad es un requerimiento huérfano que tarde o temprano alguien borrará o modificará sin entender sus consecuencias.

El tutorial no agota el tema. Quedan fuera, deliberadamente, cuestiones como la elicitación participativa de requerimientos, las técnicas de priorización avanzada, las metodologías ágiles aplicadas a la especificación, los casos de uso UML como notación detallada y los métodos formales para sistemas críticos. Cada una merecería un tutorial propio, y la bibliografía citada ofrece puertas de entrada a todos esos temas. Lo que sí esperamos haber logrado es que el lector se quede con una intuición sólida del concepto central, con el vocabulario suficiente para conversar con autores y colegas más experimentados, y con la convicción de que dedicar tiempo a redactar requerimientos funcionales de calidad no es perder tiempo: es invertirlo donde más rinde.

Si en algún proyecto futuro el lector se encuentra discutiendo con su equipo qué debe hacer

un sistema, y descubre que esa discusión es ordenada, productiva y respetuosa con las distintas perspectivas en juego, parte del mérito de esa conversación civilizada provendrá de haber asumido en serio que un requerimiento funcional es algo más que una frase en un documento. Es, dicho con sencillez, la forma técnica que adopta una promesa.

9. Bibliografía

Bourque, P., & Fairley, R. E. (Eds.). (2014). *Guide to the Software Engineering Body of Knowledge, Version 3.0 (SWEBOK Guide V3.0)* [Producido por la IEEE Computer Society. Editado por Pierre Bourque (École de technologie supérieure) y Richard E. Fairley (Software and Systems Engineering Associates)]. IEEE Computer Society. https://www.computer.org/education/bodies-of-knowledge/software-engineering

Celi-Párraga, R. J., Boné-Andrade, M. F., Mora-Olivero, A. P., & Sarmiento-Saavedra, J. C. (2023, abril). *Ingeniería del Software I: Requerimientos y Modelado del Software* (1.ª ed.) [Texto evaluado por pares doble ciego externos. Licencia Creative Commons CC BY-NC-SA 4.0. Autores afiliados a la Universidad Técnica Luis Vargas Torres de Esmeraldas, Sede Santo Domingo de los Tsáchilas (Ecuador)]. Editorial Grupo AEA.

Contreras Bernal, C. A., & Hernández Ruiz, C. M. (2022). Ingeniería de requerimientos, camino a un proyecto exitoso. *Revista Tecnología, Investigación y Academia (TIA)*, *10*(2), 90-115.

Dick, J., Hull, E., & Jackson, K. (2017). *Requirements Engineering* (4.ª ed.) [Originally published in the Springer "Practitioner Series". Corrected publication November 2017. eBook ISBN: 978-3-319-61073-3]. Springer International Publishing AG. https://doi.org/10.1007/978-3-319-61073-3

Laplante, P. A. (2017). *Requirements Engineering for Software and Systems* (3.ª ed.). CRC Press, Auerbach Publications.

Meyer, B., Bruel, J.-M., Ebersold, S., Galinier, F., & Naumchev, A. (2019, julio). *The Anatomy of Software Requirements* (Technical Report (Draft)) (Version 3, 11 July 2019. arXiv preprint). Innopolis University y IRIT, University of Toulouse.

Pressman, R. S. (2002). *Ingeniería del Software: Un Enfoque Práctico* (5.ª ed.) [Adaptación europea por Darrel Ince. Traducción de *Software Engineering: A Practitioner's Approach, 5th Edition, European Adaptation* (ISBN 0-07-709677-0), McGraw-Hill, 2001]. McGraw-Hill Interamericana de España.

Robertson, S., & Robertson, J. (2012). *Mastering the Requirements Process: Getting Requirements Right* (3.ª ed.) [Reference book for the Volere requirements methodology]. Addison-Wesley Professional.

Sommerville, I. (2011). *Ingeniería de Software* (9.ª ed.) [Traducción de *Software Engineering, 9th Edition*, Addison-Wesley, 2011]. Pearson Educación.

Wiegers, K., & Beatty, J. (2013). *Software Requirements* (3.ª ed.). Microsoft Press.