Requisitos

Jordi Pradel Miquel
Jose Raya Martos

PID_00171154

Universitat Oberta de Catalunya
www.uoc.edu

Índice

Introducción

Como hemos visto en el módulo "Introducción a la ingeniería del software", los requisitos expresan qué necesidades debe cubrir el sistema que desarrollaremos y qué restricciones ha de satisfacer. La identificación y gestión de requisitos es, por lo tanto, una de las actividades más importantes en todo proyecto de desarrollo. Al fin y al cabo, si los requisitos no se han identificado correctamente, el software no hará lo que debe hacer o no lo hará como le corresponde.

En este módulo, veremos cómo los requisitos son la herramienta básica de comunicación entre el grupo de personas que quiere que se desarrolle el sistema y el grupo de personas que lo debe desarrollar y cómo, por lo tanto, la identificación de los requisitos es una tarea compartida entre los dos grupos. Veremos también algunas de las técnicas habituales para identificar estos requisitos, que nos ayudarán a superar las dificultades ya mencionadas en el módulo "Introducción a la ingeniería del software", y también algunas técnicas para decidir cuáles son los requisitos más prioritarios.

A continuación, estudiaremos la documentación de requisitos y qué calidades debe tener. ¿Cómo sabemos que es suficientemente buena? Debemos tener en cuenta que el sistema de información que desarrollamos será tan bueno como lo sean los requisitos a partir de los cuales lo hemos desarrollado y, por lo tanto, la calidad de los requisitos afectará en gran medida a la calidad final de nuestra solución.

Finalmente, propondremos dos maneras diferentes de documentar los requisitos, cada una asociada a un tipo diferente de método de desarrollo y, por lo tanto, adecuada a unos contextos determinados: las historias de usuario, más adecuadas para procesos ágiles y menos formales, y los casos de uso, asociados al estándar UML, en los que nos centraremos.

Objetivos

Los objetivos que el estudiante debe alcanzar una vez trabajados los contenidos de este módulo son:

1. Saber identificar los requisitos candidatos de un sistema de información.

2. Saber seleccionar los requisitos del producto de software que hay que desarrollar.

3. Saber identificar similitudes y diferencias entre tres técnicas de documentación de requisitos: el estándar IEEE-830, las historias de usuario y los casos de uso.

4. Saber documentar la funcionalidad de un producto de software usando la técnica de los casos de uso.

1. Introducción a los requisitos

1.1. ¿Qué son los requisitos?

Como hemos comentado en el módulo 1, los requisitos "expresan las necesidades y restricciones que afectan a un producto de software que contribuye a la solución de un problema del mundo real" y nos sirven para delimitar qué posibles soluciones son adecuadas para el problema (las que cumplen los requisitos) y cuáles no.

¿*Requisito* o *requerimiento*?

No debemos confundirnos. Estamos hablando de requisitos, no de requerimientos.

Según el DRAE, un requisito es una circunstancia o condición necesaria para algo, mientras que un requerimiento es la acción y efecto de requerir.

Para poder determinar si una solución cumple o no los requisitos, es necesario que el requisito haga referencia a alguna característica observable del sistema, dado que, si no podemos hacer la observación de una característica, no podremos determinar el cumplimiento del requisito.

Ejemplo de característica observable

La característica "el sistema debe ser cómodo de usar" no es una característica observable, en el sentido de que su cumplimiento es subjetivo. Por lo tanto, sería necesario encontrar la manera de decir lo mismo haciendo referencia a los hechos observables de nuestro sistema como "se hará un estudio de satisfacción sobre una muestra de cincuenta usuarios y deberá obtener una nota mínima de 4 sobre 5".

Para que una característica observable sea un requisito, es necesario que exprese alguna necesidad o restricción que afecte al software.

Ejemplo de característica que no interesa a nadie

Por ejemplo, "el sistema siempre tardará más de treinta minutos en cargar la página principal del campus" es un hecho observable, pero no habrá nadie que esté interesado en que el sistema que queremos desarrollar lo cumpla; en todo caso, pretenderemos lo contrario, que el sistema tarde menos de *N* segundos en cargar la página principal del campus.

Un requisito es una característica observable del sistema que satisface una necesidad o expresa una restricción que afecta al software que estamos desarrollando.

1.2. Los *stakeholders*

Si los requisitos expresan necesidades y restricciones, ¿quiénes tienen estas necesidades? ¿Quién decide las restricciones que debe cumplir el software?

Los *stakeholders* de un proyecto son aquellas personas y entidades que tienen algún impacto o interés en éste.

De entrada, es importante distinguir entre los *stakeholders* y los usuarios. A pesar de que los usuarios siempre son *stakeholders*, no todos los *stakeholders* son usuarios. Esta distinción es importante, dado que a la hora de determinar cuáles son los requisitos de un sistema, hay que tener en cuenta todos los *stakeholders* y no sólo a los usuarios.

Los *stakeholders* de una aplicación web

Tomemos como ejemplo una aplicación web. Desde el punto de vista de los usuarios, es indiferente en qué lenguaje se ha programado (Java, C#, PHP, etc.). Por lo tanto, podríamos decir que el lenguaje de programación no es un requisito del sistema más allá del hecho de que sea web.

En cambio, desde el punto de vista de los administradores de sistemas, esta característica es muy importante, hasta el punto de que, en muchas organizaciones, los administradores de sistemas limitan las posibles plataformas de explotación a un conjunto estandarizado (aquellas que saben administrar) y, por lo tanto, limitan los lenguajes de programación a aquellos que se pueden usar con aquellas plataformas. Así, en este caso la característica anterior deviene un requisito del sistema, a pesar de que a los usuarios les resulte indiferente.

Los requisitos nos dicen qué es lo que los diferentes *stakeholders* esperan del nuevo sistema.

Por lo tanto, la función principal de los requisitos es la de comunicar las necesidades y los objetivos de los *stakeholders* a los desarrolladores: los requisitos nos dicen qué condiciones o capacidades debe satisfacer el sistema para ser valioso y útil a los *stakeholders*.

1.3. Tipo de requisitos

Podemos clasificar los requisitos en dos grandes grupos: los que hacen referencia a las **necesidades** que debe satisfacer el sistema (**qué** ha de hacer) y los que expresan **restricciones** sobre el conjunto de soluciones posibles (**cómo** debe hacerlo).

Observación

Hay que tener en cuenta que cuando decimos que los requisitos hacen referencia al modo, esto no significa que los requisitos indiquen cómo debe ser el sistema internamente, sino que expresan restricciones sobre las maneras que son aceptables y las que no lo son.

Por ejemplo, el requisito "Un alumno debe poder publicar un mensaje en el foro del grupo en el que está matriculado" pertenecería al primer grupo, mientras que el requisito "El sistema estará programado en uno de los lenguajes

corporativos" pertenecería al segundo tipo. Los requisitos del primer grupo los consideramos **requisitos funcionales**, mientras que los del segundo grupo los denominamos **requisitos no funcionales**.

Requisitos del producto y del proceso

En ocasiones, también se distingue entre los requisitos del producto (aquellos que afectan al sistema que hay que desarrollar) y los requisitos del proceso (aquellos que afectan al proceso de desarrollo). Según esta clasificación, el requisito "Un alumno debe poder publicar un mensaje en el foro del grupo en el que está matriculado" sería un requisito del producto, mientras que "El sistema estará programado en uno de los lenguajes corporativos" sería un requisito del proceso.

Un ejemplo de requisito del producto no funcional podría ser: "El sistema deberá estar disponible el 99,9% del tiempo".

1.3.1. Requisitos funcionales

Los requisitos funcionales hacen referencia a la funcionalidad que debe proporcionar el sistema.

Los requisitos funcionales nos indican qué cálculos realiza el sistema, qué datos posee, cómo los manipula, etc. En general, podemos decir que los requisitos funcionales nos indican cuál es el comportamiento del sistema ante los estímulos que le llegan del exterior.

1.3.2. Requisitos no funcionales

Los requisitos no funcionales hacen referencia a restricciones sobre el conjunto posible de soluciones.

Los requisitos no funcionales suelen tener forma de restricción y suelen afectar a gran parte del sistema. Por ejemplo, si decimos que el campus virtual que estamos desarrollando debe ser transportable a otras plataformas, ello no afecta sólo al foro o a la entrega de ejercicios, sino a todo el sistema.

Otra diferencia respecto a los requisitos funcionales es que los requisitos no funcionales no incluyen comportamiento. Su finalidad es especificar criterios que evalúen la calidad general del sistema, como seguridad, rendimiento, coste, etc., más que especificar el comportamiento del sistema (esto es, como hemos dicho anteriormente, responsabilidad de los requisitos funcionales). Volviendo al ejemplo anterior, si queremos que el campus virtual sea trasladable, lo debe ser con independencia de la funcionalidad que ofrezca.

1.4. Los requisitos a lo largo del desarrollo

Hasta ahora, hemos visto que los requisitos son clave para el éxito o el fracaso de un proyecto de desarrollo de software y que, de hecho, serán los requisitos los que guiarán el resto de las actividades de desarrollo. Por ello, con independencia de cuál sea el método de desarrollo que se ha de seguir, existe una serie de actividades relacionadas con los requisitos que siempre deberemos llevar a cabo:

- **Obtención de los requisitos**: identificar cuáles son los requisitos que los diferentes *stakeholders* quieren que cumpla el sistema

- **Gestión de los requisitos**: como parte de la gestión del proyecto, hay que decidir qué requisitos son más prioritarios, identificar los contradictorios, decidir qué queremos implementar primero, etc.

- **Documentación de los requisitos**: crear un documento o un modelo con los requisitos del sistema (con independencia de si este artefacto forma parte o no de la documentación final del sistema).

- **Verificación del sistema**: comprobar si el sistema desarrollado cumple o no los requisitos.

Nota

En este módulo, no estudiaremos las diferentes técnicas relacionadas con la verificación del sistema, que se tratan en otras asignaturas.

Evidentemente, las tareas y los artefactos concretos que usaremos en un proyecto de desarrollo dependerán del método empleado. Así, por ejemplo, los métodos con ciclo de vida en cascada desarrollarán una documentación de requisitos más exhaustiva que la de los métodos ágiles.

En este módulo veremos algunas de las técnicas que se utilizan en los diferentes tipos de métodos para llevar a cabo estas actividades. Esta compilación de técnicas no es exhaustiva (hay técnicas que no veremos) y tampoco está ligada a ningún método de desarrollo concreto (por lo tanto, es poco probable que, en un proyecto concreto, debáis aplicarlas todas).

2. Obtención de los requisitos

La primera actividad relacionada con los requisitos es su obtención. El objetivo de esta actividad es obtener la lista de todos los requisitos que, idealmente, debería cumplir el sistema por desarrollar.

Denominaremos **requisitos candidatos** a aquellos requisitos obtenidos en esta primera etapa, dado que todavía no hemos decidido si los incorporaremos o no al conjunto de requisitos de nuestro sistema. Por ejemplo, debemos tener presente que los requisitos de partida pueden ser contradictorios.

Ejemplo

Juan es un estudiante de nuestra universidad y necesita, en el momento de entregar una actividad, poderla entregar en el formato de documento que le sea más conveniente. María es la profesora de la asignatura y necesita poder limitar los formatos de documento en los que se realizan las entregas para no tener que instalar infinidad de aplicaciones a su ordenador.

En este caso, tanto "El estudiante debe poder entregar la actividad en cualquier formato" como "El estudiante debe entregar la actividad en formato PDF" son requisitos del sistema, dado que tenemos un *stakeholder* que nos ha expresado esta necesidad. Obviamente, el sistema que desarrollemos no podrá cumplir los dos requisitos.

Dado que los requisitos siempre deben estar determinados por uno o más *stakeholders*, el proceso de obtención de requisitos suele realizarse en dos etapas:

- Identificar los *stakeholders* del sistema.
- Identificar los requisitos de cada uno de los *stakeholders*.

La correcta identificación de los *stakeholders* del sistema es crítica, dado que, si nos dejamos un *stakeholder*, no podemos prever cuál será su impacto sobre el proyecto de desarrollo y, en el peor de los casos, nos podría obligar a cancelar el proyecto.

Ejemplo

Por ejemplo, podría suceder que un proyecto dependiera de una subvención pública para su financiación pero que ésta añadiera condiciones, como unos mínimos de accesibilidad, para otorgar la subvención. Si no se tiene en cuenta al Estado como *stakeholder* y no se cumplen sus objetivos, el proyecto se quedará sin financiación y se deberá cancelar.

2.1. Identificación de *stakeholders*

2.1.1. *Brainstorming* o lluvia de ideas

Dado que a menudo no tenemos un punto concreto de partida, es muy habitual usar la técnica del *brainstorming* para identificar los *stakeholders* del proyecto. Esta técnica también se podrá utilizar para identificar los requisitos de un *stakeholder* concreto (sobre todo si no tenemos acceso directo a él).

Una sesión de *brainstorming* suele ser más o menos así:

1) Creación del grupo de trabajo. Se reúne un grupo que, idealmente, debería incluir a personas con roles diferentes y variados para que tengan puntos de vista diferentes y se puedan complementar. La composición del grupo dependerá de la finalidad del *brainstorming*, pero podría incluir desarrolladores, usuarios, jefes de proyecto, etc.

2) *Brainstorming* inicial. Cada participante de la reunión propone tantas ideas como es capaz de imaginar. No hay que respetar turnos y es perfectamente lícito usar las ideas de los otros participantes para proponer nuevas (por ejemplo, si estamos identificando a usuarios potenciales y alguien piensa en "estudiantes", otro puede proponer "estudiante que este semestre no está matriculado en ninguna asignatura"). La finalidad de esta etapa es conseguir el máximo de ideas posible y, por lo tanto, no se discute ni se cuestiona ninguna propuesta.

3) Organización y consolidación del conjunto inicial. Se organizan las ideas y se identifica el grado de encabalgamiento entre éstas. Aquellas que sean muy similares se pueden consolidar en una sola. Por ejemplo, si estamos identificando *stakeholders* para nuestro campus virtual, podemos ver que "quien contrata el desarrollo" y "quien pagará el desarrollo" son dos propuestas muy cercanas y las podemos consolidar en una sola, mientras que "estudiante" es una propuesta totalmente diferente de las otras dos y, por lo tanto, es un tipo de *stakeholder* diferente.

4) Refinamiento. Se describe con algo más de detalle cada una de las propuestas para conseguir la lista definitiva.

2.1.2. Modelización de roles de usuario

La modelización de roles de usuario es una técnica orientada, concretamente, a la identificación de los usuarios del sistema que, habitualmente, se aplica en forma de *brainstorming*.

La idea básica de esta técnica es que, en lugar de buscar los requisitos de todos los usuarios individuales del sistema, los podemos agrupar según su rol y asumir que todos los usuarios individuales que tienen el mismo rol ante el sistema tendrán requisitos similares.

El resultado de esta técnica es una breve descripción de las características principales de cada tipo de usuario, como con qué frecuencia usará el sistema, su nivel de conocimiento del dominio del problema, su nivel de conocimientos informáticos o para qué usarán el sistema.

Ejemplo de un rol de usuario identificado mediante modelización de roles de usuario

- Papel de usuario: estudiante.
- No es necesariamente experto en informática, a pesar de que tiene experiencia en el uso de Internet y está predispuesto a usar el campus virtual. Quiere poder plantear consultas a los profesores, comunicarse con los otros estudiantes que estén haciendo las mismas asignaturas y llevar a cabo la entrega de las actividades, todo esto de manera no presencial.

Esta técnica se puede combinar con otras, como la de Personas, que consiste en crear una persona imaginaria (con su biografía) para representar cada uno de los roles de usuario. Así, en vez de hablar de los estudiantes como cosa abstracta, pasaríamos a hablar de María. El hecho de poner nombre (e incluso cara) al rol de usuario nos ayudará a ponernos en su piel e identificar sus requisitos.

María

María nació en Gerona hace 23 años y ha trabajado como auxiliar administrativa desde los 21. Ahora se quiere poner al día con los estudios y se ha matriculado en Empresariales. Está un poco nerviosa porque es su primer año en la universidad, pero confía en que lo logrará. No sabe muy bien cómo funciona el campus virtual pero está habituada a usar el correo electrónico y las redes sociales, por lo que cree que no tendrá problemas para utilizar el sistema.

2.1.3. Representantes de los *stakeholders*

A veces, puede suceder que no tengamos acceso directo a los *stakeholders*, especialmente a los usuarios. En estos casos, necesitaremos hablar con alguien que ejerza de representante.

El representante de un *stakeholder* es aquella persona que habla en su nombre y que nos debe transmitir sus requisitos.

Obviamente, un representante de un *stakeholder* nunca es tan fiable como el propio *stakeholder* al que representa, dado que puede ser que su conocimiento de la persona a quien representa y de las necesidades de ésta sea parcial.

Uno de los puntos particularmente complicados es que el representante suele ser, él mismo, un *stakeholder*, con sus intereses y necesidades propias. Por ello, el representante puede tener tendencia a priorizar sus intereses propios ante los de la persona representada.

Además, según quién sea el representante, nos podemos encontrar con problemáticas específicas de aquel tipo de representante que habrá que tener en cuenta.

Un representante habitual de los usuarios es el responsable dentro de la organización. En este caso, hay que tener en cuenta que es posible que el responsable no sea usuario del sistema o que el uso que haga del sistema sea diferente del que hacen los usuarios que nos interesan. Por ejemplo, es probable que los coordinadores del profesorado estén mucho más interesados en obtener informes que les faciliten el seguimiento de la actividad de los profesores que no los propios profesores.

Otro caso habitual es aquel en el que los desarrolladores deciden los requisitos sin hablar con los *stakeholders*. En general, los desarrolladores son un mal representante de los *stakeholders*, dado que no sólo desconocen el uso diario del sistema, sino que, posiblemente, estarán más interesados en los aspectos tecnológicos de la solución que en las necesidades reales de las personas a quienes representan.

Apoyo al desarrollo

No obstante, existe un caso en el que los desarrolladores son un buen representante de los *stakeholders*: el desarrollo de herramientas de apoyo al desarrollo. En este caso, los desarrolladores deben saber separar su visión como desarrolladores de la herramienta de su visión como usuarios.

Los comerciales pueden actuar como representantes, especialmente en aquellos casos en los que queremos desarrollar un producto para comercializarlo y, sobre todo, si todavía no tenemos usuarios reales del sistema, dado que son quienes tienen el contacto más directo y más cercano con los clientes y los

usuarios. Sin embargo, en este caso hay que tener en cuenta que es probable que den prioridad a los requisitos que les puedan ayudar a vender el producto, más que a los que realmente sean útiles para los usuarios. Hay que recordar también que podría suceder que los comerciales tuvieran acceso a los clientes (aquellos quienes tomarán la decisión de compra) pero no a los usuarios.

Otros representantes interesantes pueden ser los formadores, el personal de soporte técnico (al fin y al cabo, su trabajo es hablar con los usuarios y explicarles cómo funciona el sistema, de modo que saben qué problemas tienen), los analistas de negocio o los expertos en el dominio.

2.2. Identificación de requisitos

2.2.1. Entrevistas y cuestionarios

Ésta es una de las técnicas más utilizadas para la obtención de requisitos. Consiste, obviamente, en entrevistarse con los *stakeholders* para obtener directamente de ellos los requisitos que tienen sobre el sistema por desarrollar.

Las claves para que la entrevista sea productiva son:

- **Elegir correctamente a los entrevistados**. Debemos asegurarnos de la variedad y representatividad de la muestra; en nuestro caso, hay que haber identificado correctamente a los *stakeholders* y asegurarnos de que aquellos a quienes entrevistamos son una muestra significativa y representativa de ellos.

- **Evitar las respuestas condicionadas**. Cualquiera de nosotros, ante la pregunta "¿Querrías que el sistema diera respuesta en menos de un segundo?", respondería que sí, dado que entendemos que si decimos que no, el sistema será muy lento. Esta respuesta, sin embargo, no significaría que realmente nuestro límite esté en un segundo (quizá estamos dispuestos a esperar dos o tres antes de desesperarnos), pero el modo como se ha formulado la pregunta ha condicionado nuestra respuesta.

- **Evitar respuestas limitadas por el conocimiento actual**. A pesar de que es muy importante tener en cuenta el conocimiento actual, a veces puede ser contraproducente, dado que nos puede limitar la capacidad de innovar y encontrar soluciones mejores que las actuales.

Ejemplo

En el campo de las aplicaciones web, por ejemplo, es habitual que los usuarios no conozcan todas las posibilidades tecnológicas de la plataforma y que, por lo tanto, no se les ocurra la posibilidad de usar determinados controles de usuario, como autocompletar una caja de texto o el uso del *drag'n'drop*.

- **Aportar información sobre el coste de las alternativas**. Si alguien nos pregunta si queremos que un sistema haga *X* y no nos dice el coste de ello, lo más probable es que la respuesta sea un *sí* con mayúsculas (¡con independencia del valor de *X*!). En cambio, si nos dicen que el coste de desarrollar *X* supone que el proyecto se retrase un año, probablemente la respuesta sea diferente.

Como alternativa (o complemento) a las entrevistas, también podemos usar **cuestionarios**. A pesar de que, debido a su naturaleza cerrada, no son demasiado adecuados para el descubrimiento de nuevos requisitos, pueden ser muy útiles para refinar los requisitos que ya hemos descubierto.

El uso de cuestionarios puede ser especialmente útil con poblaciones relativamente grandes de usuarios. Esto nos permitirá, por ejemplo, conocer el grado de importancia relativa de los diferentes requisitos.

2.2.2. Observación y prototipado

Esta técnica consiste en la observación directa de los usuarios mientras usan un sistema de información, sobre todo para detectar requisitos relacionados con la usabilidad.

Ejemplos de observación de usuarios

Por ejemplo, si vemos que un usuario debe consultar un dato, apuntarlo y luego introducirlo en otra pantalla de la aplicación, podemos pensar que le vendría bien que el sistema recordara el dato por él.

Otro ejemplo: un usuario quiere hacer una tarea concreta de manera frecuente y necesita pasar por seis opciones de menú antes de llegar a ella. En ese caso, podríamos promocionar esta tarea de modo que el usuario tuviera acceso directo a ella.

Otro ejemplo más: estamos desarrollando una aplicación para un matadero; si observamos a los usuarios en su entorno, veremos que, por cuestiones de seguridad, éstos deben llevar unos guantes protectores que hacen muy difícil el uso del teclado, lo que nos debería hacer pensar en maneras alternativas de introducir la información.

Esta observación se puede efectuar sobre un sistema ya existente, sobre una implementación parcial del sistema final (en el caso del desarrollo incremental) o sobre un prototipo.

Por lo tanto, el prototipado es una técnica que puede facilitar la observación de los usuarios. Sin embargo, también puede ser útil en otros escenarios, por ejemplo para facilitar la comunicación con los usuarios (dado que entenderán mejor un prototipo que la descripción de un requisito) o para explorar alternativas.

Lo que distingue un prototipo de una implementación parcial es que el prototipo no forma parte del producto definitivo.

Es más importante recordar que el prototipo no se debe mantener, dado que una vez obtenidos los requisitos, este código se abandona y no forma parte del producto final. Ni siquiera necesitamos usar la misma tecnología con la que implementaremos el producto final, siempre que simule correctamente la funcionalidad que queremos explorar.

Como sabemos que no deberemos mantener el código del prototipo, lo podremos desarrollar más rápidamente y con menos coste (entre otras cosas porque no es necesario que la funcionalidad esté completa) que si estuviéramos trabajando con el producto final.

2.2.3. Listas predefinidas

Otra técnica es usar listas predefinidas[1] de requisitos que nos puedan ayudar a no pasar por alto algunos requisitos. Esta técnica es especialmente útil para los requisitos no funcionales, puesto que, como hemos visto anteriormente, éstos son independientes del comportamiento del sistema.

(1)En inglés, *checklists*.

Ejemplo

El estándar IEEE 830 nos recuerda que siempre debemos documentar, además de los requisitos funcionales, los requisitos no funcionales de lo siguiente:

- Rendimiento (volumen de usuarios, volumen de datos, etc.)
- Requisitos lógicos de la base de datos (tipo de acceso, frecuencia de los accesos, entidades, relaciones y restricciones de integridad, etc.)
- Restricciones de diseño (otros estándares, limitaciones de hardware, etc.)
- Otros atributos:
  - fiabilidad
  - disponibilidad
  - seguridad (uso de criptografía, informes sobre las acciones de los usuarios, etc.)
  - mantenibilidad
  - portabilidad

A continuación, veremos con más detalle los diferentes tipos de requisitos no funcionales que menciona la plantilla Volere, que incluye una lista predefinida de requisitos no funcionales.

Web recomendada

Volere Requirements Specification Template. http://www.volere.co.uk/template.htm (Última visita: septiembre 2010)

Requisitos de presentación

Los requisitos de presentación hacen referencia al aspecto visual del sistema por desarrollar y deben tener en cuenta aspectos como la imagen corporativa de la organización para la que estamos desarrollando el software.

Ejemplo

Como ejemplo de requisito de presentación podríamos considerar "El producto debe ser atractivo para una audiencia adolescente. Se llevará a cabo un estudio en el que se pondrá el producto al alcance de una muestra de adolescentes y se observará si empiezan o no a usar el producto antes de cuatro minutos".

Requisitos de usabilidad y humanidad

Los requisitos de usabilidad y humanidad son los requisitos relacionados con el hecho de que el sistema sea ergonómico y usable. La usabilidad del producto dependerá de las capacidades de sus usuarios y de la complejidad del producto. Algunos de los aspectos que hay que tener en cuenta en este apartado son:

- **Eficiencia de uso**: rapidez y precisión con la que el usuario usa el producto.

  Ejemplo

  Un usuario sin entrenamiento debe ser capaz de encontrar el plan de estudios de una asignatura de la que se acaba de matricular en menos de cinco minutos.

- **Facilidad de memorización**: qué volumen de información debe memorizar un usuario no habitual del sistema.

  Ejemplo

  Los menús no pueden tener más de nueva opciones y no pueden descender más de tres niveles.

- **Tasa de errores**: cuántas veces se puede equivocar el usuario intentando llevar a cabo alguna tarea.

  Ejemplo

  Un usuario que haya usado el sistema durante un mes sólo se equivocará el 1% de las veces que intente hacer algo en el sistema.

- **Satisfacción**: cuál es el nivel de satisfacción general con el producto una vez se ha usado.

  Ejemplo

  El 75% de los usuarios que prueben el nuevo sistema deben continuar usándolo (sin volver al sistema anterior) al cabo de un mes de haberlo iniciado.

- **Retroalimentación**: cuánta información necesita el usuario para confiar en que el producto está haciendo lo que se supone que debe hacer.

  Ejemplo

  Cuando el sistema inicie una tarea que dure más de un segundo, se mostrará un indicador de progreso.

En este ámbito, también podemos encontrar requisitos relativos a la personalización del producto o la internacionalización y localización de éste.

Internacionalización y localización

Según la Wikipedia, la **internacionalización** es el proceso de diseñar software de tal manera que se pueda adaptar a diferentes idiomas y regiones sin la necesidad de hacer cambios en el código del programa. La **localización** es el proceso de adaptar el software para una región específica.

También debemos considerar requisitos relacionados con la curva de aprendizaje del producto o, dicho de otro modo, qué inversión en tiempo deben emplear los usuarios antes de poder usar el sistema con confianza y productividad.

Otra categoría de requisitos que nos plantea la plantilla en esta área son los denominados de *comprensibilidad*, que hacen referencia a cómo es de intuitivo el sistema y cómo encaja en su visión del mundo (por ejemplo, usar un conjunto de símbolos conocido o seguir una guía de estilo concreta para la interfaz gráfica de usuario).

Finalmente, no podemos dejar de lado los requisitos de accesibilidad. En este sentido, hay estándares como la iniciativa de accesibilidad web[2] que nos pueden guiar sobre qué aspectos de nuestro sistema debemos tener en cuenta a la hora de garantizar el acceso a personas discapacitadas.

(2)En inglés, *web accessibility initiative*.

Requisitos de cumplimiento

En esta categoría encontramos los requisitos relacionados con la manera de cumplir las responsabilidades del sistema, como la velocidad y latencia (especialmente críticos en los sistemas de tiempo real), que hacen referencia a la velocidad con la que el sistema reacciona ante un acontecimiento.

Ejemplo

Cualquier acción del usuario debe provocar una respuesta visible antes de tres segundos.

También debemos considerar los requisitos de seguridad en el sentido del daño que podemos provocar a otras personas, ya sea porque el software esté controlando algún elemento físico que pueda causar un daño o porque dé instrucciones equivocadas a la persona. A menudo estos requisitos estarán recogidos en algún tipo de estándar nacional.

Los requisitos de precisión hacen referencia a la precisión de la información proporcionada por el sistema, como el número de decimales que se usarán para las conversiones de importes monetarios o qué sistema de unidades usará.

La fiabilidad se puede medir como el tiempo admisible entre dos caídas del sistema, mientras que la disponibilidad es el porcentaje de tiempo en el que el sistema está disponible.

Ejemplo

El sistema no puede caer más de dos veces en el mismo día.

El sistema debe estar disponible el 99% del tiempo.

La robustez nos habla de la capacidad del sistema para continuar funcionando en el caso de encontrarse en circunstancias inesperadas.

Ley de Murphy

La famosa ley de Murphy debe su nombre al ingeniero aeroespacial Edward A. Murphy y originalmente hacía referencia al hecho de que hay que diseñar los sistemas teniendo en cuenta que, si algo puede ir mal, se debe diseñar el sistema teniendo en cuenta que irá mal.

Otra categoría que se debe tener en cuenta es la de los requisitos relacionados con la capacidad (volúmenes de usuarios, de datos, etc.).

Ejemplo

La aplicación del campus virtual debe soportar la conexión simultánea de al menos el 30% de los estudiantes el primer día de curso a las 9 de la mañana.

El efecto 2000

A finales del siglo XX se produjo el denominado *efecto 2000*, que consistía, básicamente, en un error a la hora de calcular las fechas que se habían almacenando con dos dígitos, suponiendo que los dos primeros eran 19. Este efecto se debió, en parte, a errores en la previsión de la vida útil de los sistemas.

Los requisitos de escalabilidad o extensibilidad hacen referencia a cómo se espera que crezcan estos volúmenes a lo largo del tiempo.

Ejemplo

La aplicación del campus virtual debe soportar un volumen de 10.000 estudiantes el primer año de funcionamiento, que serán 15.000 al cabo de dos años.

Finalmente, consideraremos en esta categoría los requisitos de longevidad: cuánto tiempo se espera que el sistema esté en funcionamiento.

Ejemplo

La vida útil mínima de este sistema se espera que sea de diez años.

Requisitos operacionales y de entorno

En este grupo encontramos los requisitos relativos a la manera o al entorno en el que se usará el producto. Esto puede incluir requisitos relativos al modo como los usuarios deben interactuar con el sistema, a cómo debe interactuar el sistema con otros sistemas externos (qué interfaces se han de usar), a cómo se debe distribuir o a cómo se han de programar las entregas.

Ejemplo

El sistema debe poder ser usado por una persona mientras conduce un vehículo (y, por lo tanto, lo ha de poder controlar sin usar las manos).

El sistema lo debe poder usar una persona que lleve unos guantes de seguridad de malla de acero.

El sistema ha de permitir el intercambio de movimientos bancarios según el estándar Norma 43 del Banco de España.

Una persona sin conocimientos informáticos debe poder instalar el sistema fácilmente.

La aplicación se distribuirá en formato de código fuente y los usuarios deberán compilarlo para su plataforma.

Se entregará una versión nueva cada tres meses.

Requisitos de mantenimiento y apoyo

Una vez que el sistema esté en explotación, se deberá, por un lado, mantener (corregir errores y añadir funcionalidades nuevas) y, por otro, dar apoyo a los usuarios. Por lo tanto, se deben tener en cuenta las necesidades de las personas que se han de encargar de estas tareas.

Ejemplo

En caso de error, el sistema asignará un código al usuario, que el personal de apoyo podrá consultar para obtener un registro detallado de los motivos del error.

El sistema ofrecerá la posibilidad de enviar, de manera automática, un mensaje al personal de apoyo con indicaciones sobre qué estaba haciendo en este momento. Este mensaje incluirá información sobre el error que el usuario no verá.

La aplicación debe ser portable y aprovechar la misma base de código para las versiones de Windows, Linux y MacOS.

Requisitos de seguridad

En este apartado debemos considerar los diferentes aspectos de la seguridad, como el control de acceso (quién puede acceder a qué funcionalidades, en qué circunstancias lo puede hacer y sobre qué datos), la integridad o la privacidad.

Ejemplo

El profesor de un grupo de una asignatura *X* podrá acceder a todas las notas de un estudiante de su grupo en esta asignatura (no a las del resto de las asignaturas o a las de los estudiantes de otros grupos), mientras dure el semestre en el que el estudiante está matriculado de su grupo.

También entrarían en esta categoría los requisitos relacionados con la información de auditoría: qué acciones de los usuarios o acontecimientos internos del sistema deben quedar grabados y qué información se ha de grabar en cada caso.

Ejemplo

Cada vez que un usuario se intente identificar en el sistema y no lo consiga, quedará registrado en un fichero de auditoría: el nombre del usuario, la dirección desde la que se ha conectado, la fecha y la hora.

Finalmente, los requisitos de inmunidad son aquellos que nos dicen cómo se defenderá el sistema ante un ataque.

Ejemplo

Si un usuario supera el límite de cinco peticiones por segundo, su dirección quedará bloqueada y no se le permitirá hacer más peticiones al sistema durante treinta segundos.

Requisitos culturales y políticos

Los requisitos culturales son especialmente importantes cuando el equipo de desarrolladores no está familiarizado con la cultura de los usuarios potenciales del producto. Por ejemplo, podríamos recoger la manera de localizar un programa previamente internacionalizado para una cultura en concreto.

Ejemplo

Las fechas se mostrarán en formato AAAA/MM/DD.

El sistema usará el sistema métrico decimal.

En cuanto a los requisitos políticos, debemos considerar cuál puede ser la reacción de los diferentes *stakeholders* en algunas de las decisiones que se quieran tomar, como podría ser externalizar (o no) parte del desarrollo o usar (o no) componentes de software libre.

Ejemplo

Dado que la universidad quiere potenciar el uso del software libre, no se comprará ninguna licencia de ningún producto, biblioteca o bastimento para el que haya un sustituto de software libre.

La aplicación web debe funcionar con el navegador *X*, versión *Y* porque es la que tiene instalado el director general.

Requisitos legales

El último grupo de requisitos presenta las obligaciones legales del sistema que hay que desarrollar, es decir, qué leyes o qué estándares se deben cumplir.

Ejemplo

El sistema del campus virtual debe tener en cuenta el cumplimiento de la LOPD respecto al tratamiento de los datos personales de estudiantes y otros miembros de la comunidad.

2.3. Dependencias entre requisitos

A menudo, un requisito no es totalmente independiente de los otros. Puede suceder que para poder satisfacer un requisito sea necesario satisfacer antes otro. Así, por ejemplo, que los estudiantes quieran entregar las actividades en cualquier formato implica que los estudiantes entreguen actividades y no podremos implementar el requisito de los formatos si no implementamos antes el requisito de la entrega de actividades. De manera similar, un requisito podría ser un subconjunto de otro requisito; es decir, si se cumple el primero, automáticamente se cumple el segundo.

También puede suceder que el esfuerzo necesario para satisfacer un requisito dependa (de manera positiva o negativa) de si se ha satisfecho otro o no. Implementar un sistema de pago electrónico con tarjeta de débito, por ejemplo,

es más fácil si ya hemos implementado el pago con tarjeta de crédito que si no lo hemos hecho. Del mismo modo, el valor o la importancia de un requisito puede depender de otro requisito.

En cualquier caso, será importante tener en cuenta estas dependencias durante la obtención de requisitos para poderlas tener en cuenta a la hora de estimar, priorizar y seleccionar los requisitos.

3. Gestión de requisitos

Una vez obtenida la lista de requisitos candidatos, deberemos decidir cuáles son los que queremos prever. Para ello, deberemos valorar el coste de cada requisito en tiempo y recursos de desarrollo. También necesitaremos saber qué valor o importancia tiene cada requisito para los diferentes *stakeholders*. Con estos dos datos ya podremos seleccionar la lista definitiva de requisitos.

Una buena gestión de requisitos nos permitirá maximizar el valor obtenido por el proyecto de desarrollo y, por lo tanto, es una parte fundamental de la gestión del proyecto.

3.1. Valoración de requisitos

Existen muchas técnicas para valorar el coste de implementación de un requisito. La fiabilidad de éstas dependerá de la calidad de la información de partida (es más fiable, por ejemplo, si la valoramos de acuerdo con información histórica real en lugar de hacerlo con otro tipo) y del acierto que tengamos a la hora de usar esta información.

Uno de los problemas habituales durante la valoración de un requisito es la diversidad de opiniones: lo que una persona puede ver muy complicado otra lo puede ver muy sencillo. Estos conflictos suelen estar determinados por una asimetría en la información (quizá una de las dos personas conoce una cierta información que la otra desconoce o viceversa).

Para minimizar los conflictos en la estimación, es necesario que nos aseguremos de que todo el mundo trabaja con la misma información.

Pagos electrónicos

Una empresa quiere añadir la posibilidad de efectuar pagos electrónicos desde su aplicación. Para los desarrolladores, este requisito es relativamente poco costoso de implementar porque ya tienen experiencia con estos tipos de sistemas y creen que pueden aprovechar algunos componentes que tienen desarrollados para otros proyectos, por lo que creen que lo pueden tener terminado en una semana; mientras que para los clientes, en cambio, es un proceso costoso, dado que implica que deberán firmar un contrato con la empresa proveedora del sistema de pagos y este contrato deberá pasar por todo un proceso burocrático que se alargará tres meses y no será hasta entonces cuando se podrá empezar a hacer pruebas de integración. En este caso, la estimación de los desarrolladores es incorrecta porque no tienen la información que posee el cliente.

3.1.1. Unidades de valoración

En el momento de llevar a cabo la valoración del coste de un requisito, debemos hacerlo en alguna unidad. Esta unidad puede tener una contrapartida real (por ejemplo, si valoramos en horas o días de trabajo) o ser totalmente imaginaria (por ejemplo, si estimamos con "puntos").

Todas las unidades tienen sus ventajas e inconvenientes pero lo que es especialmente importante es que todo el mundo use la misma unidad y que, si queremos comparar con información previa otros proyectos u otros requisitos del mismo proyecto, ésta se encuentre en la misma unidad.

Las unidades "reales" tienen la ventaja de dar un coste inteligible. Por ejemplo, si decimos que un proyecto necesitará 2.000 horas de desarrollo, nos podemos hacer una idea de los recursos necesarios para ejecutarlo y del calendario, mientras que si decimos que necesitará 358 puntos, no podemos realizar este cálculo.

En cambio, la ventaja de las unidades "ficticias" es que nos obligan a usar información histórica real para realizar el cálculo de recursos necesarios y de calendario previsto. Así, nos facilitan ajustar las estimaciones a lo largo del proyecto. Por ejemplo, si hemos valorado un conjunto de requisitos en 5, 4 y 7 puntos, respectivamente, y nos damos cuenta de que estamos necesitando seis horas de desarrollo por punto (en lugar de las ocho que habíamos previsto), podemos calcular rápidamente los nuevos recursos y el calendario.

3.1.2. Comparación y triangulación

La técnica más básica para la estimación de un requisito consiste en hacer abstracción del trabajo requerido para implementar el requisito, encontrar uno o más que consideremos equiparables (y para los que ya tengamos una valoración) y tomar este valor como referencia de la estimación.

El principal inconveniente de esta técnica es que se necesitan valoraciones previas para poder comparar, por lo que no la podemos aplicar en las primeras estimaciones.

Para hacer la valoración más fiable, podemos complementar la comparación con la triangulación, que consiste en comparar el requisito con uno más complicado y otro más sencillo: si, por ejemplo, tenemos un requisito que hemos valorado en cuatro puntos y otro que hemos considerado como dos, un tercer requisito al que le diéramos tres puntos debería ser mayor que el de dos y menor que el de cuatro.

Utilidad de la triangulación

La triangulación es especialmente útil en el caso de usar unidades ficticias para asegurarnos de que el valor de la unidad de estimación se mantiene más o menos constante en todas las estimaciones que hacemos.

3.1.3. *Planning poker*

El *planning poker* es una técnica ágil de valoración colaborativa que recibe el nombre del hecho de que, al igual que en el juego del póquer, cada uno de los individuos realiza su "apuesta" sin saber qué piensan los otros y, al final, cada uno "muestra sus cartas".

El *planning poker* consiste en que cada uno de los participantes realiza una estimación del coste del requisito sin saber las estimaciones que han hecho los otros (en un rol o, si se quiere, con una carta de una baraja especial de *planning poker*).

Las cartas del *planning poker* tienen valores inspirados en la serie de Fibonacci para reflejar el hecho de que, a medida que la valoración es más grande, es menos precisa. Por ejemplo, tiene mucho más sentido discutir si un requisito tiene coste 1 o coste 2, que si es 21 o 22.

Una vez que todo el mundo ha realizado su valoración, "se levantan las cartas" y, si no hay consenso, se explica al resto de las partes el motivo por el que se ha hecho una estimación mayor o menor que la del resto.

Después de discutir el requisito y los motivos, se vuelve a hacer la valoración en secreto y se pone de nuevo en común. Este proceso se va repitiendo hasta que hay consenso entre todas las partes.

Lo más importante de esta técnica es que cada uno pueda hacer su estimación sin estar condicionado por las de los otros (por ello la estimación es secreta inicialmente) y que, en caso de no coincidir, se establezca un diálogo que permita asegurar que todo el mundo trabaja con la misma información.

Es importante también llegar a un consenso y, en ningún caso, caer en la tentación de hacer una media entre las diferentes valoraciones, dado que las estimaciones diferentes deben estar basadas, a la fuerza, en supuestos diferentes.

3.2. Priorización de requisitos

Una vez tenemos una idea aproximada del coste de cada requisito necesitamos calcular el valor aportado. La ventaja en este caso es que, a diferencia del coste, no necesitamos una medida absoluta del valor del requisito, sino una medida relativa al resto de los requisitos. Por ello hablamos de priorización y no de estimación del valor.

Ejemplo

Por ejemplo, en nuestro campus virtual los profesores podrían considerar más importante poder saber quién ha hecho una entrega de actividad que saber en qué formato de fichero lo ha hecho.

Por otro lado, hemos comentado anteriormente que a veces hay requisitos que entran en conflicto con otros; en este caso, necesitamos saber cuál es el que aporta más valor para que nuestro sistema lo cumpla.

Por lo tanto, parece que será relativamente sencillo priorizar los requisitos: sólo los debemos ordenar en función de su valor. Sin embargo, el problema es que el valor es relativo a los *stakeholders*, dado que el requisito, que para un *stakeholder* es muy importante, para otro puede no serlo tanto e incluso un *stakeholder* puede considerar importante incluir el requisito en un producto y otro, todo lo contrario.

Otro problema habitual es que a los *stakeholders* les cuesta priorizar los requisitos (es difícil decidir a qué requisitos se está dispuesto a renunciar). A continuación, explicamos una de las técnicas que se usan para priorizar los requisitos.

3.2.1. Votación con número limitado de votos

La técnica de los 100 dólares (Leffingwell y Widrig, 2000) consiste en dar 100 dólares imaginarios a cada uno de los *stakeholders* y decirles que los repartan entre los diferentes requisitos. Idealmente, cada *stakeholder* repartirá sus dólares entre los requisitos según el valor de cada uno. Por lo tanto, si un requisito es muy importante, le asignará muchos dólares para asegurarse de que salga elegido, mientras que, si no lo es, le asignará menos (o ninguno).

De este modo, daremos prioridad a aquellos requisitos que o bien son muy importantes para alguna de las partes (que habrá puesto una gran cantidad de dólares para asegurarse que el sistema cumple este requisito), o bien son importantes para muchas de las partes implicadas (y, por lo tanto, por acumulación de pequeños importes al final consiguen sumar un importe grande).

Uno de los problemas de esta técnica es que puede suceder que un requisito importante quede demasiado abajo en la lista porque sólo resulte valioso para un subconjunto pequeño de los *stakeholders*.

Por otro lado, habrá que asegurarse de que los *stakeholders* voten de manera honesta y, por ejemplo, eviten priorizar de menos un requisito porque saben que otros ya lo priorizarán.

3.3. Selección de requisitos

Una vez equipados con nuestra lista priorizada y estimada de requisitos, podemos proceder a la elección. Este proceso se deberá hacer para cada una de las entregas del proyecto; en el caso de las metodologías iterativas también se hará al inicio de cada iteración.

La manera más sencilla de establecer qué requisitos hay que incluir en la entrega siguiente es empezar por los más prioritarios e ir añadiendo requisitos a la lista mientras quede capacidad (tiempo y recursos) disponible para implementarlos.

Si nos encontramos con un requisito que no podemos incluir porque sería demasiado costoso, podemos decidir entre sacar alguno de la lista para hacerle sitio (a pesar de que en principio el requisito que estamos añadiendo aporta menos valor que el que estamos sacando) o pasar al siguiente en prioridad y continuar aplicando el mismo criterio hasta que no quede tiempo ni recursos disponibles.

A la hora de elegir qué requisitos hay que implementar, debemos tener en cuenta la prioridad, el coste y los recursos disponibles.

Otro factor que hay que tener en cuenta en la elección son los riesgos asociados a cada requisito; la acción por tomar dependerá de la naturaleza del riesgo y de su evolución a lo largo del tiempo.

Importancia del riesgo

El riesgo es un factor tan importante que algunos métodos de desarrollo (como UP) aconsejan implementar estos requisitos incluso antes de comprometerse a desarrollar todo el proyecto (es decir, durante la fase de elaboración del proyecto).

Por ejemplo, si debemos aplicar una tecnología con la que tenemos poca experiencia, dado que este riesgo no desaparecerá hasta que abordemos el requisito, seguramente nos interesará tratarlo lo mejor posible, de manera que si nos encontramos con problemas irresolubles la inversión realizada haya sido mínima.

En cambio, si el riesgo que hemos detectado es que consideramos poco probable que un tercero (por ejemplo, un proveedor) nos entregue un componente dentro del plazo establecido, quizá nos interese dejar este requisito para una próxima entrega, cuando el riesgo se haya reducido o haya desaparecido (por ejemplo, porque ya nos han entregado el componente).

También debemos tener en cuenta el mercado potencial del requisito (entendiendo como mercado el grupo de personas interesadas en éste). Si seleccionamos un grupo u otro de requisitos, estamos condicionando nuestro mercado

potencial; por ejemplo, si decidimos no incluir soporte para la internacionalización y localización de nuestro sistema, estamos reduciendo el mercado a un único entorno geográfico.

Por otro lado, también debemos considerar la ventana de mercado (el periodo de tiempo durante el cual nuestro producto es interesante); un mismo producto puede ser absolutamente innovador un año y no serlo en absoluto si sale al mercado dos años más tarde. Esta ventana de mercado también afectará al tipo de persona que quiera comprar nuestro producto.

Entre el mercado potencial y la ventana de mercado, hay una relación que se debe considerar: si añadimos funcionalidad, estaremos ampliando el mercado potencial, pero al mismo tiempo estaremos atrasando la introducción de nuestro producto; mientras que si extraemos funcionalidad, estaremos avanzando la introducción del producto en el mercado a cambio de reducir el tamaño de éste.

En el caso de desarrollar un sistema para la comercialización, también debemos considerar el precio que los usuarios están dispuestos a pagar por el producto en cada momento y la competencia que podríamos encontrar; muchas veces un producto inferior ha conseguido apoderarse de un mercado gracias a haber sido el primero en llegar al mercado o a haberlo hecho con un coste menor.

Finalmente, otro factor económico muy importante es el retorno de la inversión[3]. Lo podríamos definir como la medida de los ingresos que genera el uso del producto para compensar el coste de la inversión en su desarrollo.

[(3)]En inglés, *return of investment* (ROI).

Ejemplo de retorno de la inversión

Si una empresa gasta un millón de euros en un nuevo sistema informático, éste debería generar un millón de euros en beneficios (sea ahorrando costes o generando nuevos ingresos) para poder considerar la inversión como recuperada.

Quizá un requisito se considera muy importante, pero es difícil justificar, en términos económicos, su retorno y, por lo tanto, podría interesar seleccionar otro requisito que genere beneficios de manera más inmediata.

4. Documentación de los requisitos

Denominamos *especificación* al artefacto, típicamente un documento textual, que documenta el conjunto de los requisitos que se han seleccionado para el proyecto.

La especificación de requisitos presenta el contrato entre los desarrolladores y los *stakeholders* y sirve como herramienta de comunicación para los dos grupos. Por ello es un elemento central de cualquier método de desarrollo, especialmente en cuanto a la gestión del proyecto.

El estilo y la formalidad de la especificación dependerán del proyecto, pero también del contexto en el que se desarrolla y de las personas que participan en él.

Por ejemplo, el método *extreme programming* nos recomienda tener un representante de los usuarios integrado dentro del equipo de desarrollo. En este caso, el nivel de detalle y formalidad necesarios son muy bajos, dado que disponemos de una persona que puede aportar detalles y desambiguar cuando sea necesario. En este tipo de proyectos la documentación suele adquirir la forma de una lista de requisitos y para cada uno tenemos sólo aquella documentación esencial para poder retomar la comunicación verbal cuando sea necesario.

En el otro extremo, hay empresas que elaboran la especificación de requisitos en un país y subcontratan la implementación a otra empresa en otro país (normalmente en otra zona horaria). En este otro caso, necesitaremos un documento mucho menos ambiguo y detallado, dado que la comunicación no es tan fluida. Es probable que en un caso así nos planteemos la utilización de un lenguaje formal de especificación, como OCL.

Lenguaje OCL

Object constraint language (OCL) es un lenguaje formal para expresar restricciones en modelos UML y, al igual que UML, lo define el Object Management Group (OMG).

Por ejemplo, si queremos decir que no puede haber dos clientes con el mismo número de teléfono, en OCL diríamos *Cliente.allInstances()->isUnique(numTelefono)*.

Otro factor que hay que tener en cuenta es el método de desarrollo que estamos usando y el coste de introducir un cambio en los requisitos. Así, no requerirán el mismo estilo de documentación de requisitos el software que controla una sonda espacial (que una vez lanzada al espacio es muy difícil, si no imposible, de actualizar), que una aplicación web en la que es relativamente sencillo desarrollar una nueva versión.

Finalmente, hay casos especiales, como cuando disponemos de una implementación del sistema que queremos desarrollar (porque tenemos una implementación de referencia o porque estamos desarrollando un sistema nuevo para sustituir otro existente). En este caso, la necesidad de documentación será menor, dado que en caso de duda podemos acudir al sistema existente para conocer su funcionamiento.

Por lo tanto, una de las tareas del ingeniero del software será evaluar cuál es la necesidad de formalidad y detalle a la hora de documentar los requisitos para un proyecto. Con esta información, el jefe del proyecto podrá decidir qué esfuerzo (y, por lo tanto, qué porcentaje del presupuesto global del proyecto) habrá que dedicar a la documentación de requisitos: deberá tener en cuenta que tan peligroso es intentar desarrollar un proyecto sin suficiente documentación de requisitos, como dedicar los recursos y el tiempo disponibles para crear una documentación innecesariamente detallada o formal.

4.1. Calidades de una buena especificación de requisitos

El estándar IEEE 830-1998, titulado "Práctica recomendada por IEEE para las especificaciones de requisitos de software", documenta, entre otras cosas, cuáles deberían que ser las calidades de una buena especificación de requisitos. A pesar de que presupone un proceso en cascada, en el que el resultado de la especificación de requisitos es un documento no ambiguo y completo, las guías que da pueden ser aplicadas también en otros ciclos de vida.

Según este estándar, una especificación de requisitos debería ser:

- **Correcta**. Una especificación de requisitos es correcta si todos los requisitos que documenta lo son realmente; es decir, si todos los requisitos enumerados son necesidades que el software debe satisfacer. Probablemente, ésta es la propiedad más difícil de comprobar, dado que, en realidad, no sabremos si la especificación es correcta hasta que el software esté funcionando y los *stakeholders* vean realmente satisfechas sus necesidades.

- **No ambigua**. La especificación de requisitos no es ambigua si cada requisito enumerado tiene una única interpretación posible. Para conseguir este objetivo hay que usar un lenguaje muy claro y aquellos términos que puedan llevar a múltiples interpretaciones deben ser definidos correctamente en un glosario. Otra aproximación es hacer especificaciones de requisitos mediante lenguajes formales. En todo caso, hay que tener muy presente que la especificación debe estar clara tanto para quien la escribe como para quien la lee y que, probablemente, estos dos grupos pueden tener un *background* diferente y entender de manera diferente un mismo lenguaje.

- **Completa**. Para que sea completa, la especificación de requisitos debe enumerar todos los requisitos de todos los *stakeholders*, definir el comportamiento del sistema especificado para todo tipo de entradas de datos y si-

tuaciones y, finalmente, etiquetar y referenciar todas las tablas, figuras y diagramas del documento. Hay que tener en cuenta que la complementación es muy difícil de lograr, dado que el número de combinaciones de situaciones y datos de entrada que permitan que el sistema se comporte de una u otra manera puede ser infinito y, por lo tanto, es fácil que nos dejamos algún caso. Además, hay que distinguir entre los requisitos que hemos seleccionado para nuestro sistema (que sí necesitamos que estén todos) y los requisitos que habíamos identificado para nuestro sistema antes de realizar la selección (que a menudo no es necesario que queden documentados o, cuando menos, no con tanto detalle).

- **Consistente**. Decimos que una especificación de requisitos es consistente si ningún subconjunto de los requisitos enumerados entra en conflicto; por lo tanto, una especificación no consistente no es realizable, dado que no podemos desarrollar ningún software que satisfaga requisitos contradictorios.

- **Requisitos etiquetados**. Cada requisito de la especificación debería ser etiquetado con información relevante para la gestión de requisitos, como puede ser la importancia y el coste. Otro factor que puede ser importante es la estabilidad, dado que algunos requisitos serán más estables que otros, en el sentido de que se espera que sufran menos cambios a lo largo del tiempo.

- **Verificable**. Una especificación es verificable si lo es cada uno de los requisitos que enumera y un requisito es verificable si existe algún proceso finito y de coste efectivo para determinar si un software satisface o no el requisito.

- **Modificable**. Consideramos que una especificación de requisitos es modificable si la estructura y la redacción permiten hacer cambios de manera fácil y consistente. Para ello, el documento debe estar muy estructurado (con su tabla de contenidos, índice, referencias cruzadas, etc.) y no ser redundante. A pesar de que la redundancia no es, por sí misma, un error, puede llevar a errores cuando se modifica la especificación de requisitos, dado que un cambio en alguna parte del documento que no actualice también lo que es redundante con lo que se ha cambiado introduciría inconsistencias.

- **Trazable**. Finalmente, una especificación es trazable si cada requisito enumerado está claramente identificado y facilita la referencia en los artefactos desarrollados a lo largo del proyecto, ya sean otros documentos (por ejemplo, de diseño), el software desarrollado, las pruebas de software, etc. La trazabilidad es especialmente importante cuando el proyecto está a medio desarrollar o ya totalmente desarrollado, dado que, si un requisito cambia, hecho que sucede a menudo, nos interesará saber qué documentos, soft-

ware, pruebas y otros materiales habrá que cambiar como consecuencia de ello.

4.2. Buenas prácticas

Pero ¿cómo conseguimos un documento que tenga todas las calidades de una buena especificación de requisitos de software? A continuación, enumeramos algunas buenas prácticas que nos pueden ayudar a lograr este objetivo:

1) Identificadores de requisitos. A menudo necesitaremos hacer referencia a un requisito en algún otro documento o en algún otro punto del mismo documento. Por ejemplo, si queremos proporcionar trazabilidad, necesitaremos poder referirnos a los requisitos desde cualquier otro artefacto. En este sentido, es muy conveniente elegir un identificador que describa de manera breve el valor más importante del requisito, más que un identificador numérico. Esto evita que acabemos hablando de manera excesivamente abstracta sobre el requisito 23.4 o el 34A y puede aumentar, además, la modificabilidad.

2) Punto de vista global. A veces, identificamos como requisitos necesidades desde el punto de vista técnico, que nos hablan de una parte por implementar del sistema más que del conjunto del sistema visto como un todo por parte de los usuarios y el resto de los *stakeholders*. Es el caso, por ejemplo, de necesidades como "la información de matrícula indicada por el alumno se debe validar y guardar en la base de datos". Hay que evitar escribir necesidades de partes del sistema desde el punto de vista del desarrollador, dado que satisfacer una de estas necesidades no aporta, por sí mismo, ningún valor a los usuarios finales y al resto de los *stakeholders* del sistema.

3) Granularidad. Conviene definir los requisitos con un nivel de granularidad adecuado, de tal manera que no tengamos una lista demasiado larga de requisitos muy detallados ni una lista muy corta de requisitos demasiado poco detallados. A pesar de que el IEEE recomienda que la especificación de requisitos sea completa, la granularidad y el detalle exactos dependen mucho las necesidades de cada equipo y del conocimiento que se tenga de estos requisitos.

En un proceso iterativo e incremental, por ejemplo, queremos que los requisitos que estamos a punto de implementar en la próxima iteración estén muy detallados y sean completos, pero no necesitaremos que lo sean los requisitos a los que les falta mucho para ser implementados. En cambio, en un proceso en cascada se pretenderá que todos los requisitos se documenten con una granularidad más fina y con un nivel de detalle mayor.

4) Interfaz gráfica de usuario. A veces damos por supuesta una interfaz gráfica de usuario, a la hora de identificar requisitos, que aún no hemos diseñado. Así, por ejemplo, indicar en un requisito que el usuario debe poder seleccionar

la asignatura de un desplegable puede suponer indicar demasiada información sobre la interfaz gráfica de usuario, sobre todo si todavía no se ha diseñado esta interfaz.

El problema de este tipo de requisitos es que presuponen una solución (en este caso, el uso de un desplegable) que, quizá, no es la mejor y que, sobre todo, no debe ser necesariamente una necesidad para el usuario. En nuestro ejemplo, puede suceder que el usuario prefiera introducir el nombre de la asignatura en un campo de texto que autocompleta o seleccionarla de una lista (en lugar de hacerlo de un desplegable).

5) Condiciones de aceptación. Una de las calidades que perseguimos es que los requisitos sean verificables. Una buena manera de asegurarlo es no olvidarse de documentar claramente, para cada requisito, las condiciones de aceptación. De este modo, estaremos apuntando el proceso finito y de coste efectivo que seguiremos para verificar cada requisito.

6) Uso de plantillas. Otra buena práctica especialmente útil para conseguir documentos de especificación de requisitos muy estructurados (y, por lo tanto, modificables) y completos es el uso de una plantilla que nos vaya indicando los apartados que hay que documentar. En este sentido, el estándar IEEE 830-1998 ya mencionado o la plantilla de especificación de requisitos Volere son dos buenos puntos de partida.

Ejemplo de uso de plantillas

La plantilla Volere nos da un índice de los contenidos que debe contener la especificación, una pequeña plantilla sobre cómo se debe documentar cada requisito y, para cada apartado del índice, una descripción del contenido, una motivación sobre su necesidad, algunos ejemplos de contenido para aquella sección y, finalmente, otras consideraciones de interés. El índice que nos propone esta plantilla es el siguiente:

Conductores del proyecto
1. Propósito del proyecto
2. Los clientes y otros *stakeholders*
3. Usuarios del producto
Restricciones del proyecto
4. Restricciones obligatorias
5. Convenciones de nombres y definiciones
6. Hechos y asunciones relevantes
Requisitos funcionales
7. Alcance del proyecto
8. Alcance del producto
9. Requisitos funcionales y de datos
Requisitos no funcionales
10. Requisitos de presentación
11. Requisitos de usabilidad y humanidad
12. Requisitos de rendimiento
13. Requisitos operacionales y de entorno
14. Requisitos de mantenimiento y soporte
15. Requisitos de seguridad
16. Requisitos culturales y políticos
17. Requisitos legales
Otras cuestiones
18. Temas abiertos
19. Soluciones prefabricadas
20. Problemas nuevos
21. Tareas
22. Migración al nuevo producto

23. Riesgos
24. Costes
25. Documentación de usuario y formación
26. Sala de espera
27. Ideas y soluciones

4.3. Historias de usuario

Las historias de usuario son la herramienta de documentación más habitual en los métodos ágiles. Los componentes básicos de una historia de usuario son:

- Una descripción escrita de la historia (sirve como recordatorio de que existe la historia y es útil para planificar, etc.).
- Una serie de conversaciones que sirven para definir y aclarar los detalles de la historia.
- Un conjunto de pruebas que documentan los detalles y que permiten determinar cuándo está completamente implementada la historia.

A veces, las historias de usuario se documentan en tarjetas de cartón y, por ello, se usa el término CCC (*card*, *conversation*, *confirmation*). No obstante, es importante entender que la tarjeta no documenta la historia, sólo la representa (los detalles están en las conversaciones y en las pruebas).

Ejemplo de historia de usuario

Un ejemplo de historia de usuario sería "Como alumno, quiero poder entregar un ejercicio por medio del campus virtual" (esto sería la descripción, lo que pondríamos a la tarjeta). Esta historia debería ir acompañada de una serie de conversaciones entre los desarrolladores y los *stakeholders* para ir perfilando detalles, como qué formatos de documento son aceptados, si los profesores pueden o no imponer un formato, si es necesario que el sistema verifique las fechas límite automáticamente, etc. Todos estos detalles, finalmente, se deberán recopilar en un conjunto de pruebas que nos han de permitir determinar si la historia se ha implementado con éxito o no:

- Verificar que si se adjunta un archivo en formato PDF, DOC, DOCX u ODT, éste se guarda y se asocia al usuario.
- Verificar que si el profesor limita los formatos, sólo se aceptan entregas en los formatos indicados por el profesor.
- Verificar que si se adjunta un archivo en un formato incorrecto, se avisa al usuario.
- Verificar que si se intenta hacer la entrega fuera de las fechas de la actividad, se muestra un error al usuario.
- Verificar que si el usuario sale sin adjuntar el archivo, el sistema le avisa y le pide confirmación.
- Verificar que sólo se puede hacer la entrega en aquellas asignaturas en las que está matriculado el usuario.

La principal ventaja y, al mismo tiempo, el principal inconveniente de las historias de usuario es que están muy enfocadas a la comunicación verbal. Esto permite que la comunicación sea más ágil y más fluida, con lo que los requisitos serán más cercanos a las necesidades reales de los *stakeholders*, pero en cambio no quedan registradas, con lo que no tenemos todos los detalles por escrito.

No obstante, hay que tener en cuenta que la conclusión final de las conversaciones sí queda documentada y además en un formato que facilita la validación, ya que están documentadas como pruebas.

FitNesse y Cucumber

Hay herramientas en el mercado como FitNesse y Cucumber que facilitan la ejecución automatizada de pruebas a partir de su documentación.

Otra ventaja de las historias de usuario es que están escritas en el lenguaje de los usuarios o *stakeholders*. De hecho, son los usuarios o los propios *stakeholders* los que, idealmente, deberían escribir las historias de usuario.

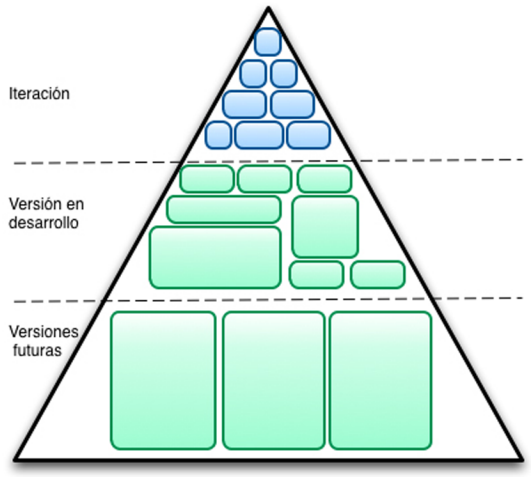
La pila de producto[4] es la lista total de historias de usuario de un proyecto de desarrollo. Dentro de este *backlog* podemos encontrar historias muy grandes (denominadas *Epics*), como "Habrá un mecanismo de registro de evaluación continua", o historias muy pequeñas, como "Un profesor puede limitar los formatos de entrega de una actividad".

(4)En inglés, *product backlog*.

La pila de producto ideal debe tener una forma denominada *de iceberg*: la parte de arriba son las historias que tendremos en cuenta en la iteración actual (y, por lo tanto, como la punta de un iceberg, las que son "visibles"). Estas historias son más pequeñas y detalladas.

La base del iceberg (la parte "no visible") son historias más grandes, menos detalladas, que aún no implementaremos y que, más adelante, habrá que dividir en historias más pequeñas para poderlas implementar.

Medida de las historias de usuario en la pila de producto



Las historias de usuario son muy útiles para los requisitos funcionales pero también nos pueden ser útiles para los requisitos no funcionales.

Historias de usuario y requisitos no funcionales

Como director técnico de la empresa, quiero que otras aplicaciones puedan acceder a la base de datos en el futuro.

Como profesor, quiero conocer la fecha de entrega con precisión de minutos.

Como director técnico de la universidad, quiero que la aplicación funcione sobre cualquier sistema operativo de esta lista: Linux, Windows, BSD.

5. Casos de uso

Los casos de uso son una técnica de documentación de requisitos muy extendida, entre otros motivos porque UML le da apoyo. Se trata de un enfoque a la manera de documentar requisitos que permite usar varios grados de detalle y de formalismo, lo que los hace adecuados en escenarios muy variados.

Ved también

Podéis encontrar más información sobre el estándar UML en los módulos "Introducción a la ingeniería del software" y "Análisis UML".

5.1. ¿Qué es un caso de uso?

> Un caso de uso "[...] es una descripción de un conjunto de secuencias de acciones, incluyendo variantes, que ejecuta un sistema para producir un resultado observable y de valor para un actor".
>
> VV. AA. (2010). *Unified Modeling Language™ (UML®)*. Object Management Group

Cockburn define los casos de uso desde el punto de vista de su función.

> "Un caso de uso captura un contrato entre los *stakeholders* de un sistema en cuanto a su comportamiento [...]. El sistema responde [...] protegiendo los intereses de los *stakeholders*".
>
> Alistair Cockburn (2001). *Writing Effective Use Cases*. Addison-Wesley

Por lo tanto, podemos decir que un caso de uso recoge el contrato entre el sistema y los *stakeholders* mediante la descripción del comportamiento observable del sistema.

Al describir un caso de uso hay un *stakeholder* que es especial (al que denominaremos **actor principal**), que es quien quiere usar el sistema para satisfacer un objetivo concreto. El caso de uso nos describe cuál es el comportamiento observable del sistema durante esta ejecución del caso de uso, de manera que podamos asegurar que se cumple el objetivo del actor principal teniendo en cuenta los intereses del resto de los *stakeholders*.

Sistema informático, actor principal

Se podría dar el caso de que el actor principal no fuera un *stakeholder*, sino un sistema informático. Por ejemplo, si estamos desarrollando un servidor de correo IMAP sin interfaz gráfica, el actor principal de la mayoría de los casos de uso es una aplicación cliente de correo, pero el *stakeholder* sería la persona que usa la aplicación cliente.

Una característica interesante de los casos de uso es que, dado que no estamos hablando del comportamiento interno del sistema, podemos describir su funcionalidad sin necesidad de describir cómo se implementa el comportamiento.

¿Qué aspecto tiene un caso de uso? A continuación ponemos un ejemplo.

Caso de uso: leer un mensaje del foro (nivel usuario)

Un usuario pide leer un mensaje del foro y el sistema le muestra el tema y el contenido. El sistema también graba que el usuario ha leído el mensaje.

Si el usuario quiere escribir una respuesta, el sistema le pedirá el tema y el contenido de la respuesta y grabará un mensaje asociado al mensaje original con este tema y contenido y la fecha actual.

La descripción del caso de uso anterior es muy informal y poco detallada. En algunos casos necesitaremos documentar de manera más detallada el caso de uso. A continuación podemos ver el mismo caso de uso documentado con un formato más adecuado a estos casos. De momento no nos preocuparemos por saber qué significa cada uno de los apartados, ya que lo veremos más adelante.

CU001: leer un mensaje del foro.
Actor principal: usuario.
Ámbito: campus virtual.
Nivel de objetivo: usuario.
Usuario: quiere leer el mensaje.
Profesor responsable del aula: quiere leer el mensaje y saber qué estudiantes lo han leído.
***Stakeholders* e intereses:**
Precondición: el usuario se debe haber identificado en el sistema.
Garantías mínimas: el sistema grabará el intento de lectura del mensaje.
Garantías en caso de éxito: el sistema mostrará al usuario el contenido del mensaje y grabará la lectura.
Escenario principal de éxito:
1. El usuario indica qué mensaje quiere leer.
2. El sistema graba el intento de lectura del mensaje por parte del usuario.
3. El sistema valida que el usuario tenga acceso al foro.
4. El sistema muestra el tema y el contenido del mensaje.
5. El sistema graba que el usuario ha leído el mensaje.

Extensiones:
1a. El usuario apaga el ordenador.
1a1. El sistema graba la lectura del mensaje aunque el usuario quizá no lo ha visto.
5a. El usuario quiere escribir una respuesta a un mensaje.
5a1. El usuario indica que quiere escribir una respuesta a un mensaje.
5a2. El sistema pide el tema y el contenido de la respuesta, sugiriendo *RE:<Tema del mensaje al que respondemos>* como tema del mensaje.
5a3. El usuario modifica el tema si quiere e introduce el contenido de la respuesta.
5a4. El sistema graba la respuesta con el tema y contenido introducidos, asigna la fecha actual como fecha de publicación del mensaje y la asocia al mensaje original.
5a3a. El usuario introduce un tema o una respuesta vacía.
5a3a1. El sistema indica que no se puede dejar vacío el tema ni la respuesta y volvemos a su punto 2a3.
5b. La base de datos no está disponible (error de servidor).
5b1. El sistema indica al usuario que no ha sido posible recuperar el mensaje y le pide que lo vuelva a probar pasado un rato.
5c. El mensaje tiene documentos adjuntos y el usuario quiere bajar uno.
5c1. El usuario indica qué documento adjunto quiere bajar.
5c2. El sistema baja el documento adjunto al ordenador del usuario.

Como podemos ver, a pesar de que el concepto de caso de uso es muy sencillo, su descripción se puede complicar bastante. Por ello, es habitual combinar varios estilos de descripción de los casos de uso en un proyecto y dejar este nivel tan detallado para los casos de uso más complicados.

5.2. Actores y *stakeholders*

Según hemos visto hasta ahora, los casos de uso están muy ligados a los *stakeholders*. Hemos mencionado también a los "actores", diciendo que hay un *stakeholder* que actúa como actor principal del caso de uso. Esto nos podría llevar a pensar que los actores y los *stakeholders* son la misma cosa, a pesar de que esto no es cierto.

Tal como indica Cockburn (2001): "Un *stakeholder* es alguien que participa en el contrato. Un actor es alguien que tiene comportamiento". Por lo tanto, mientras que los *stakeholders* participan en el caso de uso mediante sus intereses (por ejemplo, el profesor quiere que quede registrado quién ha leído el mensaje), los actores interactúan con el sistema.

Un actor puede ser una persona, una organización o un sistema informático: puede ser cualquier cosa que tenga capacidad de interactuar con nuestro sistema y de tener un comportamiento propio.

> Más formalmente, "un actor representa un conjunto coherente de roles que los usuarios del caso de uso tienen al interactuar con éste".
>
> VV. AA. (2010). *Unified Modeling Language™* (*UML®*). Object Management Group

Por lo tanto, un actor no es una persona o una organización en concreto, sino el conjunto de roles que esta persona puede tener en relación con el caso de uso. Una misma persona, organización o sistema informático puede tener diferentes roles y, por lo tanto, aparecer como diferentes actores.

Ejemplo

Un empleado de una entidad financiera puede aparecer con el rol *Empleado* cuando se conecta al sistema informático desde el ordenador de la oficina para gestionar las cuentas de sus clientes, pero puede aparecer como rol *Cliente* si se conecta vía Internet desde su casa para gestionar sus cuentas propias. A pesar de que se trata de la misma persona, desde el punto de vista del sistema informático se trata de actores diferentes.

Mientras que todos los actores serán *stakeholders* (todos tienen algún interés en el comportamiento del sistema, ya que los afecta al interactuar en él), también puede haber *stakeholders* que no sean actores. Es importante identificar estos *stakeholders* para asegurar que el comportamiento descrito tiene en cuenta sus intereses.

Ejemplo

En nuestro ejemplo, es importante identificar las necesidades del profesor, dado que aunque éste no participe en el caso de uso se debe tener en cuenta la necesidad de que haya registro sobre quién ha leído un mensaje.

5.2.1. Actores principales y de apoyo

En el contexto de un caso de uso concreto, denominamos actor principal a aquel *stakeholder* que realiza una petición al sistema para recibir uno de los servicios y así satisfacer un objetivo.

Sin embargo, en un mismo caso de uso además del actor principal pueden aparecer uno o más actores de apoyo, también denominados *secundarios*. Éstos son actores externos al sistema que proporcionan un servicio al sistema.

Los dos casos más habituales de actores de apoyo son los sistemas informáticos externos (como un sistema para autorizar pagos con tarjeta de crédito) y los usuarios que participan en el caso de uso sin ser el actor principal (por ejemplo, un operador de una centralita de llamadas que interactúa con el sistema en nombre de una persona que llama).

Operador de una centralita

En el caso del operador de una centralita, decimos que es un actor de apoyo porque el sistema no satisface su objetivo, sino el de la persona que llama. Decimos que el actor principal (la persona que llama) es quien hace la petición al sistema a pesar de que lo haga de manera indirecta (por medio del operador).

5.3. Anatomía de un caso de uso

Anteriormente hemos visto que existen varias maneras de describir un caso de uso. En este apartado entraremos en más detalle en los diferentes elementos que forman parte del caso de uso.

5.3.1. Nombre

De entrada, hemos visto que, como mínimo, un caso de uso debe tener un nombre. Este nombre es muy importante, ya que se usará para hacer referencia al caso de uso desde cualquier artefacto a lo largo del proyecto. Desde este punto de vista, el nombre debe recoger la información más importante relativa al caso de uso: el objetivo que satisface. De este modo, sólo leyendo el nombre del caso de uso, los *stakeholders* pueden valorar cuál es el objetivo que satisface sin la necesidad de buscar el documento en el que se describe.

Cada caso de uso debe tener un nombre que indique qué consigue el actor principal en su interacción con el sistema.

Normalmente se describe el objetivo mediante una frase en forma activa que empiece con un verbo (mejor "Leer un mensaje" que "Un mensaje es leído" o "Lectura de mensaje", a pesar de que lo que es realmente importante es mantener una misma convención para todos los nombres de caso de uso de un sistema).

A veces se asocia un identificador alternativo a los casos de uso (como en el ejemplo, CU001), de manera que se haga más fácil organizarlos. Pero este identificador tiene el problema de ser poco descriptivo, por lo que es recomendable usar el nombre para identificar el caso de uso siempre que sea posible.

5.3.2. Actores

Otro elemento muy importante es documentar claramente los actores del caso de uso y cuál de ellos es el actor principal.

5.3.3. Objetivos y ámbito

Tan importante como identificar los actores es identificar sus objetivos. Pero, como hemos visto antes, no basta con los objetivos e intereses de los actores, sino que debemos tener en cuenta los de todos los *stakeholders*, con independencia de si participan o no de la interacción.

Como veremos más adelante, es muy útil clasificar estos objetivos según cómo sean de generales o de concretos, de manera que podamos realizar una descripción coherente del sistema en la que no mezclemos objetivos generales con objetivos específicos.

También habrá que documentar cuál es el ámbito considerado al describir el sistema. Como también veremos más adelante, el ámbito nos dice qué queda fuera y qué queda dentro del sistema que estamos tomando en consideración.

5.3.4. Precondiciones y garantías mínimas

Las precondiciones del caso de uso nos indican qué condiciones se deben dar para que se pueda llevar a cabo la interacción descrita. A diferencia de las condiciones de error (que sí que tenemos en cuenta como extensiones), los casos en los que no se cumplan estas condiciones no los tendremos en cuenta y no formarán parte de la descripción del caso de uso.

En la descripción del ejemplo también hemos añadido las garantías mínimas (lo que el sistema debe garantizar en cualquiera de los escenarios posibles) y las garantías mínimas en caso de éxito (lo que el sistema debe garantizar para considerar con éxito la interacción).

5.3.5. Escenarios

También hemos definido un escenario principal y unas extensiones. Como podemos ver, un caso de uso puede acabar de varias maneras. Cada una de estas posibilidades es un escenario. Así, hay un escenario (el principal) que es la secuencia de acontecimientos que espera el actor principal cuando pone en marcha la ejecución del caso de uso. El escenario principal debe ser un esce-

nario de éxito (en el que se cumplan las garantías mínimas de éxito), donde el actor principal satisface el objetivo por el que ha iniciado la interacción con el sistema.

El resto de los escenarios (sean de éxito o de error) forman el conjunto de escenarios alternativos o extensiones: son escenarios que se pueden dar, pero que no son el escenario principal. Una característica importante de las extensiones es que, al describirlas, siempre lo hacemos en referencia a la secuencia de acontecimientos del escenario principal: la extensión empieza porque, en algún paso del escenario principal, se da una cierta condición que da paso a la ejecución de la extensión.

Describimos todos los escenarios como una secuencia de acciones que pueden describir:

- **Interacciones entre el usuario y el sistema**: qué hace el usuario en este punto de la interacción ("El usuario indica el tema y el contenido del mensaje") o qué respuesta da el sistema ("El sistema muestra las prácticas para entregar este mes").

- **Validaciones por parte del sistema**: "El sistema valida que el usuario tenga acceso al foro".

- **Cambios en el estado interno del sistema**: "El sistema graba que el usuario ha leído el mensaje".

Las frases que escribimos para describir las acciones deben ser sencillas y mostrar claramente cuál es la acción que lleva a cabo el usuario o el sistema. Debe estar claro quién ha de llevar a cabo la acción descrita (el usuario o el sistema) y, habitualmente, ignoran la tecnología para no limitar las posibilidades de implementación.

Además, conviene mostrar sólo aquellas acciones relevantes que hacen avanzar el proceso y describirlas como objetivos e intenciones y no de "movimientos".

Ejemplo

Así, por ejemplo, en lugar de usar 4 pasos para describir que "El sistema pide el nombre", "el usuario introduce el nombre", "el sistema pide el apellido" o "el usuario introduce el apellido", preferiremos un único paso "el usuario introduce el nombre y el apellido", puesto que esta frase recoge el hecho esencial de que el usuario proporciona nombre y apellido.

5.4. Clasificación de casos de uso

Podemos clasificar los casos de uso según dos grandes ejes: el nivel en el que se describen sus objetivos y el ámbito que se considera al describir el caso de uso.

5.4.1. Nivel de los objetivos

Para poder comparar los casos de uso entre sí, en primer lugar debemos asegurarnos de que sean efectivamente comparables. Por ejemplo, no tiene sentido comparar el caso de uso "Resolver una duda en el foro" (que incluye que el alumno mande un mensaje con la duda, recibir respuestas de los compañeros, quizá una respuesta del profesor, etc.) con el caso de uso "Entregar la solución de un ejercicio". Los dos casos de uso tienen un nivel de granularidad diferente: mientras que el primero es muy general, el segundo es mucho más específico.

Cockburn (2001) nos propone una primera clasificación en tres niveles:

- **Usuario** (*user goals*): son los más importantes, dado que son los objetivos concretos que los actores principales quieren conseguir al usar el sistema (por ejemplo, "Escribir un mensaje al foro" o "Entregar una práctica").

- **General** (*summary goals*): son el nivel más general y nos sirven para dar contexto y agrupar los objetivos de usuario (por ejemplo, "Resolver una duda en el foro" o "Cursar una asignatura").

- **Tarea** (*subfunctions*): son el nivel más concreto y sólo ofrecen una parte del valor que el actor espera (por ejemplo, "Adjuntar un archivo a un mensaje" o "Poner una calificación de una entrega a un alumno").

Para clasificar un objetivo dentro de su nivel podemos observar, por ejemplo, que los casos de uso más generales suelen responder a la pregunta *por qué*, mientras que los más específicos están cerca del *cómo*.

Cuando describamos un caso de uso deberemos documentar claramente cuál es el nivel del objetivo que hemos tomado en consideración. Si en un momento dado vemos que debemos concretar más, podemos descomponer un caso de uso en casos de uso más concretos.

Ejemplo de componer un caso de uso en casos de uso más concretos

Caso de uso: resolver una duda al foro.
Actor principal: el estudiante con la duda.
Actores de soporte: otros estudiantes, profesor.
Nivel: general.
Ámbito: campus virtual.
Escenario principal de éxito:
1. El estudiante con la duda escribe un nuevo mensaje en el foro.
2. Otros estudiantes del grupo leen el mensaje y las respuestas que pueda haber publicadas.
3. Algunos de estos estudiantes escriben respuestas al mensaje original o escriben respuestas a las respuestas.
4. El profesor del aula lee el mensaje original y las respuestas.
5. El profesor del aula decide que hay que intervenir con una respuesta "oficial" y escribe una respuesta al mensaje original o a alguna de las respuestas con su respuesta oficial.
6. El estudiante con la duda y el resto de los estudiantes del grupo leen el mensaje con la respuesta "oficial" del profesor.

Escenarios alternativos:
5a. El profesor decide que no hay que intervenir con una respuesta oficial y se acaba el caso de uso.
6b. Algún estudiante decide que la respuesta oficial no es satisfactoria.
6b1. El estudiante escribe un mensaje de respuesta a la respuesta del profesor pidiendo más indicaciones y volvemos al paso 2.

Si necesitamos concretar más, podemos ver que este caso de uso incluye varios pasos que podemos considerar, cada uno, un caso de uso a escala de usuario:

- Escribir un mensaje al foro.
- Leer un mensaje o respuesta del foro.
- Escribir una respuesta a un mensaje del foro.

El caso de uso "Leer un mensaje del foro" nos resulta familiar, dado que lo hemos descrito anteriormente. En este caso, vemos que su nivel es de usuario, ya que cumple un objetivo concreto de un usuario: leer un mensaje del foro. Sin embargo, podemos ver uno de los posibles contextos en los que puede tener lugar este caso de uso: como parte del caso de uso general de comentar una duda al foro.

Respecto a los casos de uso de tarea, cada uno de los pasos de los escenarios descritos en el caso de uso "Leer un mensaje al foro" lo podríamos considerar un caso de uso de tarea y especificarlo individualmente, a pesar de que no es habitual documentarlos.

5.4.2. Ámbito

Cuando describimos un caso de uso es importante tener claro cuál es el ámbito de la descripción que estamos haciendo: qué queda dentro del sistema que estamos describiendo, como elemento documentado, y qué queda fuera, como elemento externo.

Podemos identificar tres grandes tipos de ámbito:

- **Ámbito de organización**: estamos modelizando como caso de uso el comportamiento de la organización como un todo o de una de sus unidades, por ejemplo un departamento. Todos los sistemas (informáticos o no) y las personas de dentro de la organización que define el ámbito forman

parte del sistema que documentamos y, por lo tanto, no aparecen como actores. En cambio, otras personas, organizaciones o sistemas con quienes interactúa la organización analizada serán actores.

- **Ámbito de sistema**: estamos modelizando un sistema informático (normalmente el que queremos desarrollar). Todos los componentes (de software o hardware) del sistema son internos y, por lo tanto, no aparecen como actores.

- **Ámbito de subsistema**: estamos modelizando una parte del sistema informático, como un componente. Los otros componentes y todos los usuarios directos del componente serán actores en nuestro modelo. Este ámbito sólo es relevante cuando queremos dejar explícitamente fuera de estudio una parte del sistema.

Todos los casos de uso que hemos visto hasta ahora tenían ámbito de sistema. Veamos un ejemplo de un caso de uso con ámbito de organización.

Un caso de uso (bastante simplificado) de matrícula

Caso de uso: matrícula.
Actor principal: estudiante.
Nivel: general.
Ámbito: universidad (organización).
Escenario principal de éxito:
1. El estudiante elige las asignaturas de las que se quiere matricular y el grupo de cada asignatura.
2. La universidad confirma que la selección es correcta e indica el precio de matrícula.
3. El estudiante efectúa el pago de la matrícula.
4. La universidad registra la matrícula del estudiante y da confirmación al estudiante.

Escenarios alternativos:
2a. La selección de asignaturas no es correcta (porque el estudiante no cumple los requisitos necesarios).
2a1. El estudiante rehace su selección y volvemos al paso 2.
4b. El pago no se efectúa dentro del límite establecido por la universidad.
4b1. La matrícula se registra como errónea y se ofrece al estudiante una moratoria de 30 días.
4b2. Si el estudiante realiza el pago correctamente dentro del nuevo límite, vamos al paso 4. En caso contrario, la matrícula queda anulada.

Este caso de uso tiene como ámbito la universidad, que incluye el personal que trabaja en ella y todos los sistemas informáticos de los que dispone. Así, el paso 2 puede ser un proceso automático realizado por un sistema informático, pero también podría incluir intervención por parte del personal de la universidad en el caso, por ejemplo, que un comité tome decisiones sobre aceptación de matrícula en casos especiales.

Normalmente encontraremos cierta relación entre el ámbito de un caso de uso y el nivel del objetivo de su actor principal: cuando describimos casos de uso con ámbito de organización, tendemos a usar objetivos de nivel más general, mientras que los casos de uso con ámbitos más concretos tienden a describir objetivos de más bajo nivel.

5.5. Identificación y descripción de casos de uso

Una de las tareas más complicadas en la creación del modelo de casos de uso es la propia identificación de los casos de uso. En este apartado veremos un enfoque sencillo basado en la identificación de los actores principales, sus objetivos y la clasificación de estos objetivos en función de su nivel de generalidad.

5.5.1. Identificación de actores y objetivos

Denominamos *actor iniciador de un caso de uso* al actor que, en un momento dado, lleva a cabo una acción (manda un mensaje, pulsa un botón, selecciona una opción de un menú) que provoca la ejecución del caso de uso.

A pesar de que habitualmente es el actor principal quien pone en marcha el caso de uso (el actor principal y el iniciador son el mismo), hay algunos casos en los que no es así, como cuando el usuario del sistema actúa en nombre de algún otro o cuando el escenario se pone en marcha como resultado de un acontecimiento temporal.

En el supuesto de que el usuario actúe en nombre de otra persona, por ejemplo del operador del centro telefónico, debemos tener en cuenta que, mientras que el usuario del sistema es un *stakeholder* válido (obviamente tendrá sus intereses sobre el sistema), el actor principal es la persona que ha llamado por teléfono, dado que es la persona que quiere conseguir un objetivo.

En el caso de los casos de uso que se ejecutan de manera automatizada como respuesta a un acontecimiento temporal, se suele usar un actor virtual denominado *Reloj*, a pesar de que, obviamente, el reloj no tiene ningún interés en ejecutar el caso de uso. En este caso, hay que identificar al actor principal, que será aquel *stakeholder* que está interesado en que el caso de uso se ejecute.

Una vez identificados los actores principales (sean o no usuarios directos del sistema), podemos enumerar, para cada uno de ellos, cuáles son los objetivos que quieren obtener del sistema. Para cada uno de los objetivos que encontramos, habrá que tener un caso de uso que describa la interacción del actor principal con el sistema para satisfacer el objetivo.

5.5.2. Escenarios alternativos y extensiones

A pesar de que el estudio de las extensiones de un caso de uso puede parecer secundario (al fin y al cabo, lo que es importante es ver cómo el actor principal satisface su objetivo), es muy habitual descubrir nuevos casos de uso y nuevos requisitos mientras se están explorando las posibles extensiones de un caso de uso.

Ejemplo

Supongamos que estamos identificando casos de uso para un sistema de ventas y estamos explorando el caso de uso "Comprar productos". Al llegar al pago puede suceder que el cliente pague en efectivo, con tarjeta y con transferencia, puede ser que inicie el pago con transferencia pero que la transferencia no llegue nunca (en este caso necesitamos un caso de uso para detectar estos pedidos y tratarlos adecuadamente) o que llegue la transferencia y no cuadre el importe (necesitamos un caso de uso para notificar al cliente el problema) y así con infinidad de escenarios que, quizá, al pensar en el caso de uso "Comprar productos", no nos habrían venido a la cabeza.

5.5.3. Relaciones entre casos de uso: inclusión

A veces nos encontramos con que cuando documentamos los escenarios, de un caso de uso nos puede resultar útil para hacer referencia a otro caso de uso. Cuando esto ocurre se crea una dependencia de inclusión del primer caso de uso hacia el caso de uso que éste menciona. Se trata de una dependencia, ya que para entender el primer caso de uso habrá que leer también la documentación de los casos de uso que éste incluye.

Ejemplo

Hacer entrega de una práctica (nivel usuario)

Un estudiante quiere hacer entrega de una práctica que ha hecho. El estudiante indica la asignatura y la práctica y adjunta el archivo con la solución.

Adjuntar un archivo (nivel tarea)

El usuario selecciona el archivo de su ordenador e indica un nombre (por defecto el nombre local). El archivo se envía al sistema, que guarda el archivo, el nombre indicado, quién lo ha subido y la fecha.

En este ejemplo decimos que el caso de uso "Hacer entrega de una práctica" incluye el caso de uso "Adjuntar un archivo", dado que en el escenario del primero indicamos que se usa el segundo caso de uso.

Uso del subrayado

Es habitual el uso del subrayado para indicar, en el escenario de un caso de uso, qué acciones se corresponden a casos de uso incluidos.

Inclusión por reutilización

El escenario más típico de inclusión es aquel en el que encontramos una serie de pasos que son comunes a más de un caso de uso. Por ejemplo, el caso de uso "Escribir un nuevo mensaje en el foro" y el caso de uso "Escribir una respuesta a un mensaje" comparten la parte de redacción del mensaje en la que el usuario indica el tema y el contenido del mensaje y, opcionalmente, adjunta uno o más archivos.

En estos casos, podemos extraer la parte común en un nuevo caso de uso y establecer una relación de inclusión entre éstos: los dos casos de uso del ejemplo incluirían el caso de uso "Redactar mensaje".

En este escenario, cuando un caso de uso sea incluido en otro, lo será, como mínimo, en un tercero (así, en nuestro ejemplo, "Redactar mensaje" es incluido por "Escribir un nuevo mensaje en el foro" y "Escribir una respuesta a un mensaje").

Inclusión por descomposición

Como hemos dicho, a veces queremos descomponer un caso de uso de objetivo general en casos de uso de objetivos más concretos para añadir detalle y concreción a la documentación. En este caso, el caso de uso más general incluirá los casos de uso más concretos que habremos documentado.

Éste ha sido el uso que se ha dado a la inclusión en el ejemplo del subapartado 00, a pesar de que en éste todavía no se había explicado el significado de los subrayados.

Separación de una extensión en un caso de uso incluido

Otro caso típico de uso de inclusión se produce cuando aparece un caso de uso con una extensión muy larga. A menudo las extensiones de un caso de uso implican variar un par de pasos y volver al escenario principal, pero a veces nos encontramos con extensiones que implican una serie larga de pasos o que pueden tener extensiones dentro de la extensión. En este supuesto, se hace difícil congregar toda la variabilidad de escenarios del caso de uso original y, por ello, es recomendable separar la extensión en un caso de uso independiente.

En el ejemplo detallado del subapartado anterior "Leer un mensaje del foro", nos encontramos, por ejemplo, con la extensión "El usuario responde al mensaje", que es igual de larga o más que el escenario principal: éste es un buen candidato para separarse como caso de uso independiente.

Ejemplo

Caso de uso: leer un mensaje del foro.
Actor principal: usuario.
Ámbito: campus virtual.
Nivel de objetivo: usuario.
***Stakeholders* e intereses:**
Usuario: quiere leer el mensaje.
Profesor responsable del aula: quiere leer el mensaje y saber qué estudiantes lo han leído.
Precondición: el usuario se debe haber identificado en el sistema.
Garantías mínimas: el sistema registrará el intento de lectura del mensaje.
Garantías en caso de éxito: el sistema mostrará al usuario el contenido del mensaje y registrará su lectura.
Escenario principal de éxito:
1. El usuario indica qué mensaje quiere leer.
2. El sistema registra el intento de lectura del mensaje por parte del usuario.
3. El sistema valida que el usuario tenga acceso al foro.
4. El sistema muestra el tema y el contenido del mensaje.
5. El sistema registra que el usuario ha leído el mensaje.

Extensiones:
1a. El usuario cierra el ordenador.
1a1. El sistema registra la lectura del mensaje aunque el usuario quizá no lo ha visto.
5a. El usuario quiere escribir una respuesta a un mensaje.
El usuario **responde al mensaje en el foro**
5b. La base de datos no está disponible (error de servidor).
5b1. El sistema indica al usuario que no ha sido posible recuperar el mensaje y le pide que lo vuelva a probar pasado un rato.
5c. El usuario quiere bajar archivos adjuntos.
5c1. El usuario indica qué documento adjunto quiere bajar.
5c2. El sistema baja el documento adjunto al ordenador del usuario.
5c3. Si el usuario quiere bajar más documentos, volvemos al paso 5c1.

Caso de uso: responder a un mensaje del foro.
Actor principal: usuario.
Ámbito: campus virtual.
Nivel de objetivo: usuario.
***Stakeholders* e intereses:**
Usuario: quiere responder a un mensaje que ha leído.
Profesor responsable del aula: quiere hacer el seguimiento de los debates que se producen en su aula.
Precondición: el usuario se debe haber identificado en el sistema.
Garantías mínimas: el sistema registrará la respuesta al mensaje.
Garantías en caso de éxito: el sistema registrará la respuesta al mensaje.
Escenario principal de éxito:
1. El usuario indica que quiere escribir una respuesta a un mensaje.
2. El sistema pide el tema y el contenido de la respuesta, sugiriendo *RE:<Tema del mensaje al que respondemos>* como tema del mensaje.
3. El usuario modifica el tema si quiere e introduce el contenido de la respuesta.
4. El sistema registra la respuesta con el tema y contenido introducidos, asigna la fecha actual como fecha de publicación del mensaje y la asocia al mensaje original.

Extensiones:
3a. El usuario introduce un tema o una respuesta vacíos.
3a1. El sistema indica que no se puede dejar vacío el tema ni la respuesta y volvemos al punto 3.

5.5.4. Relaciones entre casos de uso: extensión

Originalmente había una práctica bastante extendida que consistía en no modificar los documentos de análisis una vez finalizados y, en lugar de ello, escribir documentos adicionales que indicaran los cambios que se habían producido en los requisitos.

Esto propició que se introdujera una nueva relación entre casos de uso diferente de la de inclusión y especialmente pensada para documentar extensiones complejas (como la de "Responder a un mensaje del foro").

La diferencia, cuando se utiliza una relación de extensión, es que el caso de uso que contiene el escenario principal no se modifica: no se menciona la extensión en ningún momento. En lugar de ello, nos encontramos con el caso de uso que documenta la extensión e indica cómo y dónde se añade la extensión al caso de uso original.

Supongamos, por ejemplo, que no podemos modificar los documentos de análisis y que, al describir el caso de uso "Leer un mensaje del foro", no hemos tenido en cuenta la opción de responder. El documento de partida sería el siguiente.

Ejemple

En el ejemplo "Leer un mensaje del foro", si documentamos "El usuario quiere escribir una respuesta a un mensaje" mediante una relación de extensión, el resultado es el siguiente:

Caso de uso: leer un mensaje del foro.
Actor principal: usuario.
Ámbito: campus virtual.
Nivel de objetivo: usuario.
***Stakeholders* e intereses:**
Usuario: quiere leer el mensaje.
Profesor responsable del aula: quiere leer el mensaje y saber qué estudiantes lo han leído.
Precondición: el usuario se debe haber identificado en el sistema.
Garantías mínimas: el sistema registrará el intento de lectura del mensaje.
Garantías en caso de éxito: el sistema mostrará al usuario el contenido del mensaje y registrará su lectura.
Escenario principal de éxito:
1. El usuario indica qué mensaje quiere leer.
2. El sistema registra el intento de lectura del mensaje por parte del usuario.
3. El sistema valida que el usuario tenga acceso al foro.
4. El sistema muestra el tema y el contenido del mensaje.
5. El sistema registra que el usuario ha leído el mensaje.

Extensiones:
1a. El usuario cierra el ordenador.
1a1. El sistema graba la lectura del mensaje aunque el usuario quizá no lo ha visto.
5a. La base de datos no está disponible (error de servidor).
5a1. El sistema indica al usuario que no ha sido posible recuperar el mensaje y le pide que lo vuelva a probar pasado un rato.
5b. El mensaje tiene documentos adjuntos.
5b1. El usuario indica qué documento adjunto quiere bajar.
5b2. El sistema baja el documento adjunto al ordenador del usuario.

En este ejemplo, podríamos añadir la opción de responder mediante la relación de extensión y, por lo tanto, sin modificar el caso de uso original.

Ejemplo

Caso de uso: responder a un mensaje del foro.
Extiende: el caso de uso "Leer un mensaje del foro" en su punto 5.
Actor principal: usuario.
Ámbito: campus virtual.
Nivel de objetivo: usuario.
***Stakeholders* e intereses:**
Usuario: quiere leer el mensaje.
Profesor responsable del aula: quiere leer el mensaje y saber qué estudiantes lo han leído.
Precondición: el usuario se debe haber identificado en el sistema.
Garantías mínimas: el sistema registrará el intento de lectura del mensaje.
Garantías en caso de éxito: el sistema mostrará al usuario el contenido del mensaje y registrará su lectura.
Escenario principal de éxito:
1. El usuario indica que quiere escribir una respuesta a un mensaje.
2. El sistema pide el tema y el contenido de la respuesta, sugiriendo *RE:<Tema del mensaje al que respondemos>* como tema del mensaje.
3. El usuario modifica el tema si quiere e introduce el contenido de la respuesta.
4. El sistema graba la respuesta con el tema y contenido introducidos, asigna la fecha actual como fecha de publicación del mensaje y la asocia al mensaje original.

Extensiones:
3a. El usuario introduce un tema o una respuesta vacíos.
3a1. El sistema indica que no se puede dejar vacío el tema ni la respuesta y volvemos a su punto 3.

Cuando un caso de uso (como "Responder a un mensaje del foro") extiende otro ("Leer un mensaje del foro"), existe una dependencia del primero hacia el segundo, dado que el segundo no menciona la extensión en ninguna parte y, por lo tanto, no depende de él.

A diferencia de la inclusión, es muy extraño que un caso de uso sea extensión de dos casos de uso diferentes: sería mucha casualidad que tuviéramos dos casos de uso de partida con la misma variación.

Por lo tanto, ¿qué relación entre casos de uso es más adecuada para una situación dada? Según Alistair Cockburn: "Como regla general, si un caso de uso *A* menciona un caso de uso *B*, entonces *A* incluye *B*. Casi siempre es más fácil y más claro usar inclusión".

5.6. Casos especiales

5.6.1. Autenticación de usuarios

El de la autenticación de usuarios es un caso que genera una cierta controversia. Por un lado, se puede considerar un requisito funcional y, por lo tanto, documentar el comportamiento del sistema durante la autenticación como un caso de uso. Por otro lado, se puede considerar un requisito no funcional, como un detalle necesario para conseguir la seguridad del sistema con independencia de la funcionalidad del sistema.

Algo evidente es que identificarse como usuario no es un objetivo de nivel de usuario, en el sentido de que cuando alguien se identifica en el sistema lo hace con otro objetivo (comprar un producto, entregar un ejercicio, etc.). Por lo tanto, queda claro que, en el caso de ser un caso de uso, la autenticación de usuarios sería de nivel tarea y no de nivel usuario.

Teniendo en cuenta esto, en el momento de documentar la necesidad de autenticación, debemos decidir si lo hacemos como precondición o como un paso más del escenario.

Elegiremos la primera opción (precondición) cuando no queremos que sea posible iniciar el caso de uso sin estar identificado (por ejemplo, el caso de uso "Entregar un ejercicio" requiere que el usuario esté identificado antes de empezar el caso de uso).

En cambio, elegiremos la segunda opción (del escenario) cuando queramos indicar, dentro de la secuencia de acontecimientos del escenario, qué parte se puede llevar a cabo sin estar identificado y en qué momento es necesario que el usuario esté identificado. Por ejemplo, es muy típico que las aplicaciones de comercio electrónico nos dejen hacer pedidos sin estar identificados en el sistema y sólo nos obliguen a identificarnos en el momento de efectuar el pago.

5.6.2. Alta de usuario

Otro caso especial relacionado con la identificación de usuarios es el registro o alta de usuarios. A diferencia de la identificación de usuarios, el alta puede ser un objetivo en sí misma (tiene sentido pensar que alguien accede a nuestro sistema sólo para darse de alta y, una vez se haya dado de alta, ya no haga nada más hasta otro día).

En los sistemas en los que los usuarios se dan de alta ellos mismos es habitual que este caso de uso se pueda ejecutar de manera independiente o, para facilitar las cosas, como extensión del caso de uso de identificación. Esto se nos hace un poco extraño, puesto que tenemos un caso de uso de nivel usuario que es una extensión de un caso de uso de nivel tarea. Pues bien, deberemos convivir con este hecho.

Como conclusión, podemos decir que este caso de uso se halla en un nivel intermedio entre el nivel usuario y el nivel tarea, de manera que lo podemos documentar como caso de uso independiente de nivel usuario o como caso de uso de nivel tarea (ya sea también independiente o como extensión del caso de uso de identificación) dentro de otro caso de uso de nivel usuario.

5.6.3. Mantenimientos (CRUD)

Cuando identificamos los casos de uso de un sistema solemos encontrarnos con una serie elevada de casos de uso del tipo "Crear un *X*", "Consultar un *X*", "Actualizar un *X*" y "Borrar un *X*". Éstos son lo que denominamos *mantenimientos* o casos de uso CRUD[5] (crear-recuperar-actualizar-borrar).

(5)Del inglés, *create-read-update-delete*.

Según lo que hemos visto hasta ahora, cada uno de estos pequeños casos de uso es un caso de uso de nivel de usuario, dado que su actor principal tiene un objetivo claro que espera lograr mediante aquel caso de uso.

Ahora bien, identificar y documentar cada mantenimiento como un conjunto de cuatro o cinco casos de uso a escala de usuario puede ser, muchas veces, una tarea repetitiva, sin valor y propensa a errores, ya que todos los casos de uso de creación se asemejarán mucho entre sí, todos los de consulta también, etc.

Por lo tanto, una opción totalmente válida es crear, en estos casos, un único caso de uso de nivel general que aglutine los diferentes objetivos de usuario del mantenimiento en un único caso de uso. Esto tiene la ventaja de simplificar el trabajo y evitar la repetición innecesaria.

Ejemplo (incompleto) de mantenimiento de campus, edificios y aulas

Caso de uso: gestionar campus, edificios y aulas.
Actor principal: un administrador del sistema.
Nivel: general.
Escenario principal de éxito:
1. El usuario selecciona una consulta de campus, edificios y aulas.
2. El sistema muestra una lista de los campus del sistema.
3. El usuario selecciona un campus.
4. El sistema muestra una lista de los edificios del campus.
5. El usuario selecciona un edificio.
6. El sistema muestra una lista de las aulas del edificio.
7. El usuario vuelve al paso 1, 3 o 5 hasta que queda satisfecho (y el caso de uso se acaba).

Escenarios alternativos:
a. Crear una nueva aula.
   1. El usuario indica que quiere crear una nueva aula en el edificio seleccionado.
   2. El usuario indica el nombre del aula.
   3. El sistema registra la nueva aula y vuelve al punto 6.

b. Borrar un aula.
   1. El usuario indica que quiere borrar un aula.
   2. El sistema verifica que el aula no está en uso en el calendario del semestre actual.
   3. El sistema marca el aula como eliminada (a pesar de que continuará apareciendo en los horarios de los semestres pasados) y vuelve al punto 6.

c. Cambiar el nombre de un aula
   1. El usuario indica que quiere cambiar el nombre de un aula.
   3. El usuario indica el nuevo nombre del aula.
   4. El sistema registra el cambio de nombre y vuelve al punto 6.

d. Crear un nuevo edificio
   1. El usuario indica que quiere crear un nuevo edificio en el campus seleccionado.
   3. El usuario indica el nombre y la dirección del edificio.
   4. El sistema graba el nuevo edificio y vuelve al paso 4.

Como inconveniente, la agrupación de los mantenimientos en casos de uso de nivel general impide detallar información concreta de cada objetivo de nivel más bajo. Así, por ejemplo, si queremos documentar los permisos que cada actor tiene sobre cada operación, deberemos hacerlo en un documento aparte.

Un enfoque propuesto por Cockburn (2001) consiste en empezar usando casos de uso de mantenimiento de nivel general y, sólo si descubrimos que debemos dar más detalles o identificar de manera individual alguna parte del mantenimiento, extraer casos de uso de nivel más bajo y documentarlos aparte. En tal caso, podemos usar una relación de inclusión para indicar, en el caso de uso más general, en qué caso de uso más detallado se describe la parte que hemos extraído.

5.6.4. Casos de uso parametrizados

A menudo nos encontraremos con la obligación de escribir varios casos de uso muy parecidos entre sí. Es el caso, por ejemplo, de muchos mantenimientos ya mencionados y otros casos de uso en los que identificamos un patrón común que después se aplica a varios casos de uso.

El caso de uso de ejemplo de mantenimiento que aparece en el subapartado anterior es bastante específico, ya que agrupa los mantenimientos de campus, edificios y aulas. Probablemente, otros muchos mantenimientos serán sencillos, del estilo "Gestión de personas", "Gestión de asignaturas", "Gestión de departamentos", etc.

Probablemente, todos estos casos de uso seguirán una misma estructura. Por ejemplo, que son del tipo siguiente.

Ejemplo

Caso de uso: gestionar asignaturas.
Actor principal: un administrador del sistema.
Nivel: general.
Escenario principal de éxito:
1. El usuario pide hacer una busca de asignaturas.
2. El usuario introduce el nombre o fragmento de nombre de la asignatura.
3. El sistema muestra una lista de las asignaturas coincidentes (todas, si no se ha introducido ningún nombre de asignatura).
4. Opcionalmente, el usuario cambia el criterio de ordenación (nombre o número de créditos) y vuelve al paso 3.
5. Los pasos 2 a 4 se repiten hasta que el usuario queda satisfecho (y el caso de uso termina).

Escenarios alternativos:
a. Crear una nueva asignatura
   1. El usuario indica que quiere crear una nueva asignatura.
   2. El usuario indica nombre y número de créditos de la asignatura.
   3. El sistema registra la nueva asignatura y vuelve al punto 3.

b. Borrar una asignatura
   1. El usuario indica que quiere borrar una asignatura.
   2. El sistema verifica que la asignatura no se imparte en el semestre actual.
   3. El sistema marca la asignatura como eliminada (a pesar de que continuará apareciendo en los horarios de los semestres pasados) y vuelve al punto 3.

c. Actualizar una asignatura
   1. El usuario indica que quiere actualizar una asignatura.
   2. El usuario indica el nombre o número de créditos.
   3. El sistema graba el cambio de nombre o de número de créditos y vuelve al punto 3.

En este caso de uso podemos observar que, probablemente, todos los mantenimientos auxiliares serán muy parecidos en estructura, a pesar de que en cada uno variará:

- De qué hacemos el mantenimiento (en este caso, de asignaturas).

- Cuáles son los datos de los que hay que informar al crear una nueva instancia (en este caso, nombre y número de créditos).
- Qué datos son modificables (en este caso, nombre y número de créditos).
- Qué datos se muestran en la lista (en este caso, de nuevo, nombre y número de créditos).
- Qué datos se pueden usar para filtrar la busca (en este caso, sólo el nombre de la asignatura).
- Qué datos se pueden usar para reordenar la busca (en este caso, nombre y número de créditos).
- Qué verificaciones hay que hacer al borrar una entidad (en este caso, que la asignatura no se imparta en el semestre actual).

Escribir cada uno de estos casos de uso por separado no es sólo tedioso, sino también propiciador de error, dado que probablemente queremos que el comportamiento común de estos casos de uso sea totalmente consistente a lo largo del sistema y un error en uno de los casos de uso podría romper esta consistencia. Por otro lado, tener información duplicada también es un inconveniente a la hora de efectuar el mantenimiento de los casos de uso, ya que si queremos cambiar la estructura general de los casos de uso, deberemos cambiar muchos casos de uso diferentes.

Para resolver estos problemas podemos escribir un tipo de plantilla, un caso de uso parametrizado. Para cada elemento de la plantilla que puede variar, introducimos un parámetro.

Ejemplo

Caso de uso: gestionar entidades.
Nivel: general.
Escenario principal de éxito:
1. El usuario pide hacer una busca de entidades.
2. Opcionalmente, el usuario introduce uno o más campos de busca.
3. El sistema muestra una lista de las entidades coincidentes (todas, si no se ha introducido ningún campo de busca).
4. Opcionalmente, el usuario cambia el criterio de ordenación (indica algún campo de ordenación) y vuelve al paso 3.
5. Los pasos 2 a 4 se repiten hasta que el usuario queda satisfecho (y el caso de uso termina).

Escenarios alternativos:
a. Crear una nueva entidad.
   1. El usuario indica que quiere crear una nueva entidad.
   2. El usuario indica campos de creación de la entidad.
   3. El sistema graba la nueva entidad y vuelve al punto 3.

b. Borrar una entidad.
   1. El usuario indica que quiere borrar una entidad.
   2. El sistema verifica que la entidad se puede borrar.
   3. El sistema marca la entidad como eliminada y vuelve al punto 3.

c. Actualizar una entidad.
   1. El usuario indica que quiere actualizar una entidad.
   2. El usuario indica los cambios en los campos actualizables.
   3. El sistema registra el cambio y vuelve al punto 3.

Una vez tenemos un caso de uso plantilla, podemos definir casos de uso usando la plantilla. Para ello se debe indicar qué plantilla se usa (utilizando una relación de inclusión) y qué valores adquieren los parámetros de ésta.

Ejemplo

Caso de uso: gestionar asignaturas.
Actor principal: un administrador del sistema.
Nivel: general.
Escenario principal de éxito:
1. El usuario gestiona asignaturas usando el caso de uso Gestionar entidades, en el que:
- La entidad es *asignatura*.
- Los campos de busca son el nombre de la asignatura.
- Los campos de creación, actualizables y de ordenación, son el nombre y el número de créditos.
- La entidad se puede borrar si la asignatura no se imparte en el semestre actual.

5.6.5. Modelización basada en casos de uso de procesos de negocio

El uso más típico de modelización basada en casos de uso es el de documentar los requisitos de un sistema informático desarrollado para una organización. En este escenario, la mayoría de los casos de uso se describirán con ámbito de sistema y casi todos con el objetivo de nivel usuario.

Un uso de los casos de uso muy diferente de éste es la modelización de procesos de negocio. Se trata de modelizar mediante casos de uso un sistema que, en lugar de ser un sistema informático, es una organización que incluye sistemas informáticos, hardware, personas, etc. Por lo tanto, por definición, los casos de uso tendrán ámbito de organización.

A partir de aquí, podemos ver que los actores primarios serán aquellas personas u organizaciones que piden servicios a la organización para conseguir ciertos objetivos; cada caso de uso será un servicio que la organización ofrece a uno de estos actores primarios para satisfacer uno de sus objetivos.

Una vez elaborado un modelo de procesos de negocio basado en casos de uso, si, además, queremos analizar uno de los sistemas informáticos que forman parte de esta organización, también podemos usar casos de uso. La cuestión, entonces, es si existe alguna relación entre los casos de uso con ámbito de organización y los casos de uso con ámbito del sistema informático.

La conexión no es, en todo caso, trivial. A pesar de que podemos partir de los casos de uso de la organización para identificar casos de uso del sistema, normalmente no tendremos una conexión clara entre unos y otros para el 100% de los casos.

Por lo tanto, hay que tener en cuenta que si hacemos modelización de procesos de negocio y de uno de los sistemas informáticos de la organización, los dos basados en casos de uso, habrá que tener muy claro a cuál de los dos documentos pertenece cada caso de uso y no mezclar casos de uso de uno con los del otro.

Resumen

En este módulo nos hemos centrado en los requisitos del software, que hemos definido como características observables del sistema por desarrollar que satisfacen necesidades o expresan restricciones de aquellas personas o entidades que tienen algún impacto o interés en el proyecto y que denominamos *stakeholders*. Los requisitos se pueden clasificar como funcionales, que hacen referencia a las necesidades que deben satisfacer el sistema, o como no funcionales, que son aquellos que expresan restricciones sobre el conjunto posible de soluciones.

Para desarrollar un proyecto habrá que identificar cuáles son los *stakeholders* de nuestro sistema y obtener sus requisitos. Técnicas como el *brainstorming*, las entrevistas y los cuestionarios, la observación, el prototipado y las listas predefinidas nos ayudan a realizar una compilación de requisitos de más calidad.

A lo largo del proyecto también habrá que gestionar estos requisitos para decidir cuáles se deberán en cuenta, para lo que habrá que estimar su coste, priorizarlos y elegir para maximizar el valor obtenido para el desarrollo, teniendo en cuenta los recursos disponibles.

Los requisitos recogidos a lo largo del proyecto se documentan en lo que denominamos *especificación*, que actúa como contrato entre los desarrolladores y los *stakeholders* y como herramienta de comunicación entre ellos. Es importante que este documento tenga una gran calidad, ya que de ello depende la calidad del software que obtendremos: debemos conseguir mantener unos buenos criterios de calidad, como los mencionados por el IEEE, para lo que conviene seguir unas buenas prácticas.

En este módulo hemos visto dos estilos de documentación de requisitos. Las historias de usuario, más adecuadas a desarrollos ágiles, confían mucho en la comunicación oral entre desarrolladores y *stakeholders* y sólo documentan un título y unas pruebas de aceptación. Los casos de uso, en cambio, documentan de manera textual la interacción entre un actor y el sistema para obtener un cierto objetivo.

Actividades

1. Suponed que tenemos un caso de uso "Corregir una actividad" con la especificación textual breve siguiente:

Corregir una actividad (nivel global)

El profesor baja del campus virtual todos los documentos de entrega de la actividad entregados hasta la fecha y el sistema registra como consultadas todas las entregas disponibles también hasta entonces. El profesor corrige las entregas (fuera del sistema). El profesor introduce las calificaciones en el sistema, que comprueba que sólo introduzca calificaciones a entregas que previamente haya consultado.

Haced la especificación textual detallada, usando la plantilla detallada del subapartado 00. Cread casos de uso de nivel de usuario e incluidlos en éste para aquellas partes en las que lo creáis oportuno. Haced también la especificación detallada de estos nuevos casos de uso.

2. Indicad si los requisitos siguientes están expresados correctamente. En el supuesto de que no lo estén, reescribidlos correctamente (podéis ver el subapartado 4.2):

a) El sistema debe funcionar sobre plataforma Linux y Windows.
b) Usaremos un índice con los campos *nombre* y *descripción del producto Apache Lucene* para que los usuarios puedan buscar los productos con cualquier palabra clave.
c) La web debe funcionar sobre los navegadores más usados.
d) Usaremos el bastimento .Net Entity Framework para acceder a la base de datos.
e) Los usuarios podrán rellenar la solicitud fácilmente.

3. Supongamos que estamos desarrollando un sistema de venta de entradas para teatros por medio de Internet. Decid si las entidades y personas siguientes son o no *stakeholders*. En el supuesto de que también sean actores del sistema, indicadlo. Justificad vuestra respuesta.

a) La persona que quiere comprar la entrada.
b) La persona que atiende la centralita telefónica de atención al usuario.
c) Los propietarios de los teatros.
d) Los actores de las obras.
e) El acomodador del teatro.
f) Las personas que van al teatro pero que no han comprado directamente la entrada.
g) El personal de la taquilla del teatro.
h) Los administradores de sistemas de la empresa en los que están hospedados los servidores.
i) Los desarrolladores de la aplicación.
j) El comercial que debe convencer a los teatros de que entren en la Red y pongan sus entradas a la venta por medio de la aplicación.
k) El software que usan los teatros para gestionar las ventas de entradas en la taquilla.
l) Los desarrolladores del software que usan los teatros para gestionar las ventas de entradas en la taquilla.
m) El jefe de proyecto.

4. Suponed que estamos desarrollando un sistema para la gestión de ayudas públicas a la formación en empresas. Identificad tres roles de usuarios y dos *stakeholders* potenciales que no sean usuarios y justificad cuál es su implicación en el proyecto.

5. Suponed que os han pedido que desarrolléis una aplicación muy sencilla: se trata de permitir a los usuarios (trabajadores de una empresa) publicar anuncios de tipo clasificados ("vendo mi coche", "busco un ordenador de segunda mano", etc.) en la intranet de la empresa. No es necesario que se pueda responder, sólo publicar los anuncios (con título, texto y fecha de caducidad) y no os pueden dar más documentación ni podéis hablar con nadie más. En este caso, ¿podemos decir que tenemos representante de los *stakeholders*?

6. Supongamos ahora que, en el desarrollo del sistema anterior, han decidido que podemos enviar un cuestionario a los usuarios finales del sistema y nos han dado las preguntas siguientes. ¿Creéis que son adecuadas? Si no es así, indicad cuál es el problema y cómo se podrían mejorar.

a) ¿Creéis que se deberían poder adjuntar fotografías a los anuncios?
b) ¿Creéis que se deberían añadir más campos?
c) ¿Os parece bien que los anuncios caduquen automáticamente al cabo de un mes?
d) ¿Un usuario debería poder modificar un anuncio una vez publicado?

7. Dad dos ejemplos para cada una de las categorías de requisitos del IEEE 830 mencionadas en el subapartado 2.2.3.

8. Suponed que tres personas están haciendo una estimación mediante la técnica del *planning poker* y valoran el requisito en 2, 4 y 8 puntos. ¿Cuál es el valor que debemos usar para la estimación?

9. De los requisitos de nuestro sistema sabemos que:

- El requisito A tardaremos un día en implementarlo.
- El requisito B es casi el doble de complicado que A.
- El requisito C es el doble de complicado que B.
- El requisito D es el triple de complicado que B.
- El requisito E es el triple de complicado que C.
- El requisito F es más complicado que C pero menos que D.
- El requisito G está entre B y C.
- El requisito H está entre el doble de C y el doble de F.

¿Cuántos días tardaríamos en implementar todo el proyecto? ¿Qué sucede si nos hemos equivocado al valorar A en un 10%?

10. A continuación os presentamos un caso de uso extraído de la documentación de OpenUP. Comparad y comentad las diferencias de la plantilla empleada con la que hemos usado en el subapartado 5.1. El ejemplo completo lo podéis encontrar en EPF Wiki (última visita: octubre del 2010).

Nombre (no aparece como apartado): sacar dinero (de un cajero automático)

1. Descripción breve

Este caso de uso describe cómo el cliente del banco usa el cajero automático para sacar dinero de su cuenta.

2. Actores

Cliente del Banco
Banco

3. Precondiciones

Existe una conexión activa con el banco.
El cajero tiene efectivo disponible.

4. Flujo básico de acontecimientos

1. El caso de uso empieza cuando el Cliente del Banco inserta su tarjeta.
2. Se ejecuta el caso de uso "Validar usuario".
3. El Cajero Automático muestra las diferentes alternativas que el cliente tiene disponibles (podéis ver el requisito de soporte SR-xxx para la lista de alternativas). En este caso, el Cliente del Banco selecciona "Sacar efectivo".
4. El Cajero Automático pide qué cuenta se debe usar. Podéis ver el requisito de soporte SR-yyy para los tipos de cuenta que se deben soportar.
5. El Cliente del Banco selecciona una cuenta.
6. El Cajero Automático pide el importe.
7. El Cliente del Banco introduce el importe.
8. Se envía el ID de tarjeta, el PIN, el importe y la cuenta al banco como una transacción. El Consorcio del Banco responde con "continuar/no continuar", según si la transacción ha ido bien o no.
9. Se dispensa el efectivo.
10. Se devuelve la tarjeta.
11. Se imprime el recibo.
12. El caso de uso finaliza con éxito.

5. Flujos alternativos

5.1. Usuario incorrecto

Si en el paso 2 del flujo básico el caso de uso "Validar Usuario" no se completa con éxito, entonces:

1. El caso de uso finaliza con error.

5.2. Cuenta equivocada

Si en el paso 8 del flujo básico la cuenta seleccionada por el Cliente del Banco no está asociada con esta tarjeta, entonces:

1. El Cajero Automático mostrará el mensaje "Cuenta incorrecta; por favor, vuelva a intentarlo".
2. El caso de uso continúa en el paso 4.

[...]

5.7. No hay respuesta del Banco

Si en el paso 8 del flujo básico no hay respuesta del Banco en tres segundos, entonces:

1. El Cajero Automático lo vuelve a probar, hasta en tres ocasiones.
2. Si no hay respuesta del Banco, el Cajero Automático mostrará el mensaje "Red no disponible; pruebe de nuevo más tarde".
3. El Cajero Automático devolverá la tarjeta.
4. El Cajero Automático indicará que está "Cerrado".
5. El caso de uso finaliza con una condición de error.

[...]

6. Escenarios clave

6.1 No hay respuesta del Banco

7. Poscondiciones

7.1. Poscondiciones de éxito

El usuario ha recibido el efectivo y los registros internos se han actualizado.

7.2. Poscondiciones de error

Los registros se han actualizado como corresponde.

8. Requisitos especiales

[SpReq:WC-1] El cajero dispensará efectivos en múltiplos de 20$.
[SpReq2:WC-2] El importe máximo en un reintegro es de 500$.
[SpReq:WC-1] El cajero mantendrá un registro que incluirá fecha y hora de todas las transacciones con el banco (sean completas o incompletas).

11. Identificad un caso de uso para un sistema de alquiler público de bicicletas (como el Bicing de Barcelona) en el que el actor principal no es el usuario que alquila bicicletas.

12. Imaginad que estáis desarrollando el software que gestiona un teléfono móvil y estáis en un modo de diagnóstico en el que salen unas cifras que no sabemos qué significan. ¿Qué caso de uso puede estar relacionado con esta funcionalidad? ¿Quién sería el actor principal?

13. Suponed que pasáis por la calle y veis a un amigo vuestro concentrado en su teléfono móvil. Vuestro amigo os explica que está comprando unas entradas para el teatro y que está

buscando el botón de pagar pero no lo encuentra. Indicad objetivos globales, de usuario y de tarea en esta situación.

14. Suponed que estamos desarrollando una aplicación para la gestión de preguntas y respuestas de usuarios al estilo de Yahoo Answers o Stack Exchange. Identificad tres actores y, para cada uno de ellos, identificad como mínimo un objetivo de nivel global, uno de usuario y otro de tarea que estén relacionados entre sí. Describid de manera informal el caso de uso que permita lograr cada uno de los objetivos de nivel usuario o global identificados.

Ejercicios de autoevaluación

1. ¿Cuáles son las dos condiciones más básicas para que una cierta característica se pueda considerar un requisito?

2. ¿Qué relación existe entre el concepto de usuario y el de *stakeholder*?

3. ¿Cuál es la función principal de los requisitos?

4. ¿Qué diferencia existe entre los requisitos funcionales y los que no lo son?

5. ¿Cuáles son las principales actividades relacionadas con los requisitos que habrá que llevar a cabo a lo largo del proyecto?

6. ¿Es posible que un sistema deba satisfacer dos requisitos contradictorios?

7. Durante el *brainstorming* inicial se aportan ideas sin entrar en valorarlas, sin evitar repeticiones y sin hacer ninguna selección. ¿Cómo se llega a un resultado útil que no tenga estos problemas?

8. Indicad tres actividades relacionadas con los requisitos que se puedan hacer mediante un *brainstorming*.

9. ¿Qué técnicas nos pueden evitar la obligación de pensar en los requisitos de cada uno de los usuarios de manera individual?

10. ¿Cómo denominamos a la persona que habla en nombre de un *stakeholder* al que no tenemos acceso? ¿Qué problemas nos puede ocasionar?

11. ¿Por qué la pregunta "¿Quieres que el sistema devuelva todos los resultados paginados de 20 en 20 en menos de 1 segundo?" puede ser inadecuada para una entrevista de identificación de requisitos?

12. ¿Podemos construir el sistema final a partir de un prototipo?

13. Poned dos ejemplos de aspectos que hay que tener en cuenta en cuanto a la usabilidad y la humanidad, según Volere.

14. ¿A qué tipo de requisito de la lista Volere pertenece cada uno de estos aspectos?

- Consideraciones sobre la imagen corporativa de la organización.
- Satisfacción de los usuarios.
- Accesibilidad por parte de personas discapacitadas.
- Tiempo medio entre caídas del sistema.
- Requisitos sobre cómo se deben programar las entregas de los diferentes artefactos y versiones del sistema.
- Sistemas operativos que necesita soportar el sistema.
- Unidades que se usarán para mostrar longitudes, pesos, etc.
- Lenguajes en los que se puede mostrar la interfaz del sistema.

15. ¿Qué información deberemos tener de cada requisito candidato, antes de poder seleccionar cuáles tomaremos realmente en consideración?

16. ¿Qué dos grandes tipos de unidades podemos usar para valorar requisitos? Poned un ejemplo de cada una.

17. Si disponemos de unos cuantos requisitos ya valorados, ¿qué técnicas nos pueden ayudar a considerar el resto?

18. ¿Por qué es importante que haya consenso cuando se usa la técnica de estimación *planning poker*? ¿Por qué no resolvemos los conflictos por mayoría?

19. ¿Cuáles son las principales dificultades de priorizar los requisitos?

20. ¿Qué técnica nos puede ayudar a priorizar los requisitos cuando tenemos múltiples *stakeholders* o *stakeholders* que no tienen claro cómo se debe priorizar?

21. ¿Cuáles son las calidades de una buena especificación de requisitos según el IEEE?

22. ¿Qué defecto tiene una especificación de requisitos que se puede entender de maneras diferentes e incompatibles entre sí?

23. ¿Qué defecto tiene una especificación de requisitos en la que varios requisitos se llevan la contraria?

24. ¿Qué defecto tiene una especificación de requisitos si algunos de los requisitos son tales que requerirían una gran cantidad de dinero y tiempo para comprobar si un sistema los satisface?

25. ¿Qué defecto tiene una especificación de requisitos en la que resulta difícil introducir cambios?

26. ¿Una historia de usuario es una tarjeta con una descripción y un conjunto de pruebas de aceptación?

27. ¿Qué medida deberían tener las historias de usuario de una pila de producto?

28. ¿Qué diferencia existe entre un actor de un caso de uso y un *stakeholder*?

29. ¿Qué sucede si la precondición de un caso de uso no se cumple?

30. ¿Cuál es el escenario principal de un caso de uso?

31. ¿Qué tres niveles de objetivos de casos de uso identifica Cockburn?

32. ¿Cuál es el ámbito en el que una persona puede formar parte del sistema que estamos estudiando en lugar de aparecer como actor?

33. ¿Cuál es el ámbito en el que un componente de software del sistema que estamos estudiando se puede considerar un actor que interactúa con un subsistema del que escribimos casos de uso?

34. ¿El actor iniciador de un caso de uso debe ser el actor principal? En caso contrario, dad contraejemplos.

35. ¿Cómo podemos evitar que dos casos de uso contengan una buena parte de la documentación igual y, por lo tanto, repetida en dos documentos?

36. ¿Cómo podemos separar la documentación de una extensión muy larga del documento del caso de uso para poderle definir, a su vez, extensiones?

37. Si disponemos de la especificación textual de un caso de uso en un documento que no podemos modificar (por ejemplo, porque la política de la organización es no modificar documentos de análisis una vez publicados) y queremos añadir una extensión, ¿cómo podemos hacerlo?

38. Indicad dos maneras de modelizar la autenticación de usuarios en un modelo de casos de uso.

39. ¿Cómo podemos evitar una explosión en el número de casos de uso parecidos cuando estamos documentando mantenimientos con casos de uso del estilo CRUD?

40. ¿En qué consiste un caso de uso parametrizado?

41. ¿Qué ámbito más habitual tendrán los casos de uso si hacemos modelización de procesos de negocio?

Solucionario

Actividades

1.

Caso de uso: corregir una actividad.
Actor principal: profesor.
Ámbito: campus virtual.
Nivel de objetivo: general.
***Stakeholders* e intereses:**
Profesor: quiere corregir las entregas de la actividad.
Estudiantes: quieren que se les corrija la actividad y, por lo tanto, saber la nota.
Precondición:
El profesor se debe haber identificado en el sistema.
El estudiante debe haber entregado la actividad.
Garantías mínimas: el sistema registrará el hecho de que el profesor ha descargado los documentos.
Garantías en caso de éxito: el sistema registrará la nota de cada entrega.
Escenario principal de éxito:
1. El profesor baja los documentos de las entregas de la actividad.
2. El sistema registra el hecho de que el profesor ha descargado los documentos.
3. El profesor introduce las calificaciones de las entregas.
4. El sistema valida que el usuario haya descargado antes los documentos.
5. El sistema registra la calificación de cada una de las entregas.

Extensiones:
4a. El profesor no ha descargado el documento de la entrega.
4a1. El sistema indica al profesor que no puede aceptar calificaciones de entregas que no han sido descargadas.
4a2. Vuelven al paso 1.

Caso de uso: descargar los documentos de las entregas de una actividad.
Actor principal: profesor.
Ámbito: campus virtual.
Nivel de objetivo: usuario.
***Stakeholders* e intereses:**
Profesor: quiere descargar los documentos para corregir la entrega.
Precondición:
El profesor se debe haber identificado en el sistema.
Garantías mínimas:
Garantías en caso de éxito:
El profesor obtiene los archivos.
El sistema registrará el hecho de que el profesor ha descargado los documentos.
Escenario principal de éxito:
1. El profesor elige una actividad.
2. El sistema muestra la lista de estudiantes que han hecho la entrega.
3. El profesor pide el documento de un estudiante.
4. El sistema envía al profesor el documento de la entrega del estudiante.
5. Si el profesor quiere descargar más documentos, vuelve al paso 3.

Extensiones: –
Caso de uso: introducir calificaciones de entregas.
Actor principal: profesor.
Ámbito: campus virtual.
Nivel de objetivo: usuario.
***Stakeholders* e intereses:**
Profesor: quiere publicar las notas de las entregas.
Precondición:
El profesor se debe haber identificado en el sistema.
Garantías mínimas:
Garantías en caso de éxito: el sistema registra las notas.
Escenario principal de éxito:
1. El profesor elige una actividad.
2. El sistema muestra la lista de estudiantes que han hecho la entrega.
3. El profesor indica un estudiante y una nota.
4. El sistema registra la nota de la entrega.
5. Mientras queden notas por introducir, volvemos al paso 3.

Extensiones:
4a. El profesor no ha bajado el documento de la entrega.
4a1. El sistema indica al profesor que no puede aceptar calificaciones de entregas que no han sido bajadas.
4a2. Vuelven al paso 2.

2.

a) "El sistema debe funcionar sobre plataforma Linux y Windows" es difícil de verificar, dado que no indica qué versiones concretas de Linux y Windows se deben soportar. Una versión correcta sería: "El sistema debe funcionar sobre plataforma Linux (núcleo 2.6 y entorno gráfico Gnome 2.32) y Windows 7".
b) "Usaremos un índice con los campos *nombre y descripción del producto Apache Lucene* para que los usuarios puedan buscar los productos con cualquier palabra clave" no tiene una visión global del sistema (Lucene es un detalle de implementación que no interesa a los usuarios). La versión correcta sería "Los usuarios podrán hacer búsquedas de productos por palabras clave".
c) "La web debe funcionar sobre los navegadores más usados" es difícil de verificar. Una versión correcta sería "La web se visualizará correctamente con los navegadores Internet Explorer, Firefox, Chrome y Safari en su versión más reciente en el momento de poner el proyecto en producción y la versión inmediatamente anterior". A pesar de que no indica versiones completas, es fácil de verificar en el momento de poner el proyecto en producción.
d) "Usaremos el bastimento .Net Entity Framework para acceder a la base de datos" no es un requisito desde el punto de vista del usuario (si no es que hay algún *stakeholder* que nos lo ha pedido). En este caso, no tiene sentido reescribirlo.
e) "Los usuarios podrán rellenar la solicitud fácilmente". También es difícil de verificar. Una versión verificable: "Un usuario sin entrenamiento debe poder rellenar correctamente la solicitud sin ayuda exterior en menos de cinco minutos".

3.

a) La persona que quiere comprar la entrada es un *stakeholder*, dado que es un usuario (y, por lo tanto, un actor) del sistema.
b) La persona que atiende la centralita telefónica de atención al usuario es un *stakeholder* potencial, puesto que, para resolver las incidencias de los usuarios, es probable que necesite acceder al sistema como usuaria.
c) Los propietarios de los teatros son *stakeholders*, dado que, en última instancia, serán ellos quienes pondrán el dinero para financiar el desarrollo.
d) Los actores de las obras no son *stakeholders*, ya que no les afecta el sistema.
e) El acomodador del teatro es un *stakeholder*, dado que tiene, como mínimo, una necesidad que cubrir: que el sistema le diga cuál es el asiento que debe ocupar cada persona (aunque sea imprimiendo una entrada con el número de asiento).
f) Las personas que van al teatro pero que no han comprado directamente la entrada no son *stakeholders*, puesto que no tienen ningún interés ni interacción con el sistema.
g) El personal de la taquilla del teatro es *stakeholder*, dado que el sistema afecta a su manera de trabajar y, en todo caso, su trabajo es complementario al del nuevo sistema. Podría ser que fueran actores si usan el nuevo sistema para vender las entradas que se compran en la taquilla.
h) Los administradores de sistemas de la empresa en los están hospedados los servidores son *stakeholders*, ya que deben poder administrar el nuevo sistema.
i) Los desarrolladores de la aplicación no son *stakeholders*.
j) El comercial que debe convencer a los teatros de que entren en la Red y pongan sus entradas a la venta por medio de la aplicación es un *stakeholder*, ya que necesita que el sistema sea fácil de vender.
k) El software que usan los teatros para gestionar las ventas de entradas a la taquilla no es un *stakeholder*, a pesar de que podría ser un actor.
l) Los desarrolladores del software que usan los teatros para gestionar las ventas de entradas en la taquilla son *stakeholders*, dado que necesitan que el nuevo sistema se pueda integrar con el software que han desarrollado.
m) El jefe de proyecto no es un *stakeholder*.

4. Usuarios:

- El funcionario que gestiona las solicitudes.
- El funcionario que gestiona el presupuesto.
- La persona de la empresa que pide la ayuda.

Stakeholders:

- Los diferentes gobiernos (local, autonómico, etc.).
- Las empresas que imparten la formación.

5. En este caso, el representante de los *stakeholders* es el propio desarrollador de la aplicación.

6.

a) En "¿Creéis que se deberían poder adjuntar fotografías a los anuncios?" no se indica el coste de añadir esta funcionalidad al sistema, por lo que es difícil que nos digan que no. Una manera más correcta sería: "¿Creéis que se deberían poder adjuntar fotografías?".

Tened en cuenta que añadir esta funcionalidad implicará que no se pueda poner fecha de caducidad a los anuncios.

b) "¿Creéis que se deberían que añadir más campos?" es una pregunta demasiado abierta. Sería mejor "¿Qué campo añadiríais?" o, mejor todavía, sugerir qué campos se pueden añadir.

c) "¿Os parece bien que los anuncios caduquen automáticamente al cabo de un mes?" es una pregunta correcta para un cuestionario.

d) "¿Un usuario debería poder modificar un anuncio una vez publicado?" tiene el mismo problema que la primera pregunta: dado que no da ninguna indicación del coste, la respuesta natural a la respuesta es sí. Una posible solución: "¿Un usuario debería poder modificar un anuncio una vez publicado?". En caso afirmativo, "¿Qué funcionalidad deberíamos sacrificar para incorporar ésta?".

7.

- Rendimiento
  - El sistema debe soportar 2.000 usuarios conectados al mismo tiempo.
  - El sistema debe guardar los datos relativos a dos millones de solicitudes.

- Requisitos lógicos de la base de datos
  - El sistema debe guardar, para un pedido, qué productos se compraron, qué cantidad de cada producto y cuál era el precio.
  - La suma de los importes de los pagos de un pedido debe ser igual a la suma del valor de los productos comprados.

- Restricciones de diseño
  - El sistema debe funcionar sobre un teléfono móvil con 256 MB de RAM, procesador ARM v8 y sistema operativo Android.
  - El diseño del sistema debe seguir la compilación de buenas prácticas y patrones de la organización.

- Fiabilidad
  - En el caso de problemas con el servidor de bases de datos, en ningún caso puede quedar el sistema en un estado inconsistente.
  - El sistema debe guardar automáticamente el trabajo realizado para recuperarlo automáticamente en el caso de caída no prevista.

- Disponibilidad
  - El sistema debe poder operar de manera que, si cae un servidor, otro se encargue de continuar dando servicio a los usuarios.
  - El sistema debe poder estar disponible el 90% del tiempo.

- Seguridad
  - El sistema debe registrar los accesos a información sobre clientes realizados por los usuarios.
  - El sistema debe impedir que los usuarios de una oficina accedan a los datos personales de los clientes de otra oficina.

- Mantenibilidad
  - El sistema debe tener en cuenta que, en el futuro, se incorporarán nuevos proveedores para el servicio de mensajería y se querrá poder cambiar entre éstos fácilmente.
  - El sistema debe registrar, en el caso de error inesperado, información sobre la transacción en curso y también el punto exacto en el que se ha producido el error para facilitar el diagnóstico.

- Portabilidad
  - El sistema debe poder funcionar sobre un sistema gestor de bases de datos diferente del actual.
  - El sistema debe poder funcionar sobre los sistemas operativos Linux 2.6, Windows 7 y MacOS X 10.6.

8. Ninguno de los tres valores. Es necesario que hablen y expongan el motivo de la valoración de cada uno y, una vez que se han puesto de acuerdo, volver a dar una estimación.

9. Tardaremos 42 veces lo que tarde A, es decir, 42 días. Si nos hemos equivocado en un 10% al valorar A, pero el resto de estimaciones está bien, tardaremos un 10% más (46,2 días) en acabar.

10. La plantilla de OpenUP no tiene en cuenta los objetivos de los actores, sino sólo su comportamiento. Por otro lado, incorpora información que no aparece en la plantilla que hemos empleado en el subapartado 5.1, como los escenarios clave o los requisitos especiales.

11. Un ejemplo podría ser "Retirar una bicicleta estropeada". En este caso, el actor principal sería el encargado del mantenimiento de las bicicletas.

12. Un caso de uso relacionado con esta funcionalidad podría ser "Diagnosticar problema interno" y el actor principal, el encargado de reparar el teléfono.

13. Objetivo global: ir al teatro

Objetivo de usuario: comprar las entradas

Tarea: iniciar el pago

14. Actor: **visitante**. Es un tipo de usuario que suele llegar a la aplicación de manera casual (por ejemplo, por medio de un buscador) con un interés concreto y que, una vez haya encontrado la respuesta a la pregunta que tiene, no continuará usando el sistema. Su nivel técnico puede ser muy bajo e incluso puede ser un usuario muy ocasional de Internet. Su motivación principal para usar el sistema es encontrar una respuesta a una pregunta que tiene. Existe la posibilidad de que se registre como miembro para plantear la pregunta si no encuentra la respuesta.

Actor: miembro de la comunidad. Es un usuario que se ha registrado en el sistema para poder realizar preguntas o respuestas. Su nivel técnico no debe ser necesariamente muy alto pero está habituado a usar aplicaciones en Internet. Su motivación principal para utilizar el sistema es participar en la comunidad para ganarse un cierto prestigio, dado que no pagamos por las intervenciones.

Actor: moderador. Puede ser un miembro de la comunidad con un cierto prestigio o alguien puesto por la empresa que gestiona la aplicación. En cualquier caso, es alguien con autoridad aceptada por el resto de los miembros de la comunidad cuya finalidad es identificar las intervenciones poco apropiadas y los miembros que no se comporten según los criterios establecidos.

Actor	Nivel	Objetivo
Visitante, miembro	Global	Resolver una duda propia
Visitante, miembro	Usuario	Encontrar una respuesta
Visitante	Usuario/tarea	Registrarse como miembro
Visitante, miembro	Tarea	Leer una pregunta con sus respuestas
Miembro	Usuario	Crear una pregunta
Miembro	Tarea	Redactar la pregunta
Miembro	Tarea	Ver la lista de sus preguntas
Miembro	Global	Resolver una duda a algún otro
Miembro	Usuario	Encontrar una pregunta para intervenir en ella
Miembro	Tarea	Ver la lista de preguntas con pocas o ninguna respuesta
Miembro	Tarea	Redactar una respuesta
Moderador	Global	Moderar la comunidad
Moderador	Usuario	Encontrar intervenciones poco adecuadas
Moderador	Tarea	Ver la lista de intervenciones sospechosas

Casos de uso:

Resolver una duda propia (global)

El usuario (sea visitante o miembro) tiene una duda y accede al sistema para encontrar una respuesta. Si no encuentra la respuesta y es un miembro de la comunidad, puede crear una pregunta. Si no es miembro, puede registrarse en este momento y convertirse en miembro.

Registrarse como usuario (usuario)

Un usuario visitante decide que quiere formar parte de la comunidad y se quiere registrar. El sistema le pedirá los datos personales que formen su perfil y le asignará unas credenciales para identificarse en el futuro.

Crear una pregunta (usuario)

Un miembro de la comunidad puede crear una nueva pregunta en cualquier momento. Una vez creada la pregunta, ésta aparecerá en la lista de sus preguntas para que pueda ir fácilmente a encontrar las respuestas.

Encontrar una respuesta (usuario)

El usuario puede indicar una serie de palabras clave y el sistema le mostrará una lista de preguntas que, o bien en la pregunta o bien en la respuesta, contienen las palabras clave mencionadas. El usuario elige una y el sistema muestra la pregunta junto a todas sus respuestas. Alternativamente, el usuario puede llegar directamente a la página que muestra la pregunta, ya sea por medio de un buscador o mediante los favoritos de su navegador. Si el usuario era un miembro, también puede ver la lista de sus preguntas y llegar al detalle desde ahí.

Resolver una duda a algún otro (global)

El usuario (un miembro de la comunidad) quiere participar para resolver una duda de algún otro. Lo primero que hace es encontrar una pregunta para intervenir en ella.

Encontrar una pregunta para intervenir en ella (usuario)

El usuario puede buscar una pregunta por palabra clave (por ejemplo, para buscar preguntas sobre los temas que domina) o ver la lista de preguntas con pocas o ninguna respuesta. Una vez elegida la pregunta en la que quiere intervenir, redactará la respuesta y la enviará.

Moderar la comunidad (global)

Los moderadores son los responsables de encontrar intervenciones poco adecuadas, responder a las quejas de los usuarios y cerrar una pregunta cuando ésta ya tiene un número suficiente de respuestas.

Encontrar intervenciones poco adecuadas (usuario)

El moderador pide una lista de intervenciones sospechosas y el sistema le muestra aquellas intervenciones (sean preguntas o respuestas)

Ejercicios de autoevaluación

1. Debe ser observable y expresar una necesidad o restricción que afecte al software por desarrollar. (Podéis ver el subapartado 1.1).

2. Los usuarios siempre son *stakeholders*, ya que tienen intereses en el sistema por la misma razón que lo usarán. Pero no todos los *stakeholders* son usuarios, dado que hay personas que no usarán directamente el sistema pero tienen interés en él. (Podéis ver el subapartado 1.2).

3. Comunicar las necesidades y los objetivos de los *stakeholders*. (Podéis ver el subapartado 1.2).

4. Los requisitos funcionales expresan necesidades que ha de satisfacer el sistema, qué debe hacer, mientras que los no funcionales expresan restricciones sobre las posibles soluciones, cómo lo debe hacer. (Podéis ver el subapartado 1.3).

5. Obtención, gestión, documentación y verificación. (Podéis ver el subapartado 1.4).

6. No, ya que no sería posible satisfacerlos. Éste es uno de los motivos por los que, una vez tenemos una lista de requisitos candidatos, hay que hacer una selección de los que realmente queremos prever. (Podéis ver los apartados 2 y 3).

7. Durante la fase de organización y consolidación del conjunto inicial y la de refinamiento se filtran y documentan mejor las ideas seleccionadas. (Podéis ver el subapartado 2.1.1).

8. La identificación de *stakeholders*, la identificación de requisitos y la modelización de roles de usuario. (Podéis ver los subapartados 2.1.1 y 2.1.2).

9. La modelización de roles de usuario y las personas. (Podéis ver el subapartado 2.1.2).

10. Representante. (Podéis ver el subapartado 2.1.3).

11. Porque condiciona las respuestas (menos de un segundo, de 20 en 20) y se limita al conocimiento actual, en el sentido de que puede ser que haya soluciones alternativas a la paginación para devolver grandes cantidades de datos (por ejemplo, una barra de desplazamiento continua). (Podéis ver el subapartado 2.2.1).

12. No. Podemos usar el prototipo para resolver dudas sobre los requisitos del sistema, pero, por definición, el prototipo no se emplea como parte del producto definitivo. Si lo usáramos, ya no se trataría de un prototipo, sino de una implementación parcial. (Podéis ver el subapartado 2.2.2).

13. La lista de aspectos mencionados son: eficiencia de uso, facilidad de memorización, tasa de errores, satisfacción, retroalimentación, curva de aprendizaje, comprensibilidad y accesibilidad. (Podéis ver el subapartado "Requisitos de usabilidad y humanidad").

14.

- Consideraciones sobre la imagen corporativa de la organización (de presentación).
- Satisfacción de los usuarios (usabilidad y humanidad).
- Accesibilidad por parte de personas discapacitadas (usabilidad y humanidad).
- Tiempo medio entre caídas del sistema (consecución).
- Requisitos sobre cómo se deben programar las entregas de los diferentes artefactos y versiones del sistema (operacionales y de entorno).
- Sistemas operativos que debe soportar el sistema (mantenimiento y soporte).
- Unidades que se usarán para mostrar longitudes, pesos, etc. (culturales y políticos).
- Lenguajes en los que se puede mostrar la interfaz del sistema (culturales y políticos).

(Podéis ver el subapartado 2.2.3).

15. Necesitaremos una estimación del coste de implementarlo, así como conocer la importancia que tiene para los *stakeholders* y el riesgo que implica. También se deberá tener en cuenta los recursos disponibles, que no son información de los requisitos candidatos. (Podéis ver el apartados 3 y 3.3).

16. Reales, como las horas, o ficticias, como puntos. (Podéis ver el subapartado 3.1.1).

17. La comparación y la triangulación. (Podéis ver el subapartado 3.1.2).

18. Porque es necesario que cada uno pueda hacer su estimación sin estar condicionado; por lo tanto, tampoco debe estar condicionado por la mayoría y es importante que, en caso de duda, haya diálogo. Así se puede descubrir cuáles son los diferentes supuestos que nos han llevado a valoraciones distintas. (Podéis ver el subapartado 3.1.3).

19. Que el valor es relativo a los *stakeholders*, que puede ser que no se pongan de acuerdo y que puede suceder que a los *stakeholders* les cueste priorizar. (Podéis ver el subapartado 3.2).

20. La votación con número limitado de votos. (Podéis ver el subapartado 3.2.1).

21. Ha de ser correcta, no ambigua, completa, consistente, verificable, modificable y trazable, y debe tener los requisitos etiquetados. (Podéis ver el subapartado 4.1).

22. Es ambigua. (Podéis ver el subapartado 4.1).

23. Es inconsistente. (Podéis ver el subapartado 4.1).

24. No es verificable. (Podéis ver el subapartado 4.1).

25. No es suficientemente modificable. (Podéis ver el subapartado 4.1).

26. No, una historia de usuario es eso y además una conversación que no queda registrada. (Podéis ver el subapartado 4.3).

27. Las historias que se han de tener en cuenta en la iteración actual, las más prioritarias, deberían ser más breves y estar más detalladas. Las historias que aún no implementaremos, las menos prioritarias, deberían ser más extensas. (Podéis ver el subapartado 4.3).

28. El actor tiene comportamiento en la interacción con el sistema, interactúa con el sistema, mientras que el *stakeholder* puede no tenerlo. (Podéis ver el subapartado 5.2).

29. Si la precondición de un caso de uso no se cumple, no es posible que el caso de uso se ejecute y, por lo tanto, no se podrá llevar a cabo la interacción descrita. (Podéis ver el subapartado 5.3.4).

30. La secuencia de acontecimientos que espera el actor principal cuando pone en marcha el caso de uso y que lo llevan a satisfacer su objetivo. (Podéis ver el subapartado 5.3.5).

31. De más concretos a más generales: tarea, usuario y general. (Podéis ver el subapartado 5.4.1).

32. El de organización. (Podéis ver el subapartado 5.4.2).

33. Ámbito de subsistema. (Podéis ver el subapartado 5.4.2).

34. El actor iniciador también podría ser un actor de apoyo que interactúa con el sistema por orden del actor principal (por ejemplo, el operario de una central de llamadas) o puede ser el reloj (que usamos para representar el iniciador cuando un caso de uso se inicia en respuesta a un acontecimiento temporal). (Podéis ver el subapartado 5.5.1).

35. Podemos crear un caso de uso con la parte común a los dos e incluirlo en los dos casos de uso. (Podéis ver el subapartado "Inclusión por reutilización").

36. Podemos documentar la extensión como un nuevo caso de uso e incluirlo en el caso de uso original. (Podéis ver el subapartado "Separación de una extensión en un caso de uso incluido").

37. Podemos crear un caso de uso nuevo para la extensión y usar la relación de extensión para indicar en qué punto del caso de uso original modifica el comportamiento. (Podéis ver el subapartado 5.5.4).

38. Podemos indicar, como precondición de los casos de uso que lo requieren, que es necesario estar autenticado. Otra solución es documentar la autenticación como un paso más del caso de uso; en este segundo caso, si aquélla fuera compleja, se podría considerar un caso de uso de nivel de tarea e incluirla donde sea necesario. (Podéis ver el subapartado 5.6.1).

39. Podemos crear un único caso de uso para simplificar la documentación resultante. (Podéis ver el subapartado 5.6.3).

40. Consiste en un caso de uso que usa una plantilla que permite evitar redundancias y crea varios casos de uso que siguen la misma plantilla. (Podéis ver el subapartado 5.6.4).

41. En el caso de modelizar procesos de negocio, lo más habitual es que los casos de uso tengan ámbito de organización. (Podéis ver el subapartado 5.6.5).

Glosario

actor *m* Alguien que tiene comportamiento en un caso de uso. Puede ser una persona, pero también una organización o un sistema (de software o de otro tipo) externo al sistema que analizamos.

actor de apoyo *m* Actor externo al sistema que proporciona un servicio en el sistema.

actor iniciador *m* Actor que inicia la interacción con el sistema. Puede ser el actor principal, pero también puede ser un intermediario entre éste y el sistema. También puede ser que el caso de uso se inicie de manera automática o en reacción a un acontecimiento externo, caso en el que el actor iniciador sería el reloj o este acontecimiento, respectivamente.

actor principal *m* *Stakeholder* que efectúa una petición al sistema para recibir uno de los servicios y así satisfacer un objetivo.

ámbito (de un caso de uso) *m* Definición de qué es lo que consideramos un sistema analizado: qué queda dentro y qué queda fuera de éste. Existen tres grandes tipos de ámbitos: organización, sistema y subsistema.

ámbito de organización *m* Ámbito en el que el sistema analizado es una organización o negocio. Todos los sistemas (informáticos o no) y las personas de dentro de la organización analizada forman parte del sistema que documentamos y, por lo tanto, no aparecen como actores.

ámbito de sistema *m* Ámbito en el que el sistema analizado es un sistema informático (normalmente el que queremos desarrollar). Todos los componentes del sistema son internos y, por lo tanto, no aparecen como actores.

ámbito de subsistema *m* Ámbito en el que el sistema analizado en el caso de uso es una parte de un sistema informático, quizá un subsistema o un bastimento. Los otros componentes del sistema informático y todos los usuarios directos del componente serán actores en el modelo.

artefacto *m* En el contexto de la ingeniería del software, cada uno de los documentos, modelos, programas, etc., que se generan como resultado del trabajo del ingeniero.

autenticación *f* Acto de establecer o confirmar la autenticidad de alguien (normalmente el usuario del sistema informático); es decir, confirmar que el usuario es, efectivamente, quien dice ser. Una manera típica de autenticación es pedir al usuario una contraseña que sólo la persona que dice ser conoce.

backlog *m* Lista de historias de usuario, ya sea de todas las historias de un proyecto en desarrollo (*product backlog*) o de las que se incluyen en una iteración.

brainstorming *m* Lluvia de ideas. Técnica de trabajo en grupo diseñada para generar un gran número de ideas para solucionar un problema. En el contexto de la ingeniería del software se puede usar, por ejemplo, para identificación de *stakeholders* y para identificar requisitos, entre otros usos.

caso de uso *m* Documento que recoge el contrato entre el sistema y sus *stakeholders*. Describe el comportamiento del sistema y las interacciones en varios escenarios y muestra cómo el sistema responde a las peticiones y los objetivos de los *stakeholders*.

caso de uso de extensión *m* Caso de uso que describe una extensión de otro caso de uso (que denominamos *de base*).
Ved **extensión**

caso de uso de base *m* Caso de uso del que hay una extensión descrita en otro caso de uso (que denominamos *de extensión*).
Ved **extensión**

checklist *f* Término inglés para *lista predefinida*.

create, read, update and delete Funcionalidades básicas del almacenamiento persistente que, a menudo, se ofrecen a los usuarios de los sistemas informáticos para el mantenimiento de ciertos datos. Se pueden documentar como casos de uso individuales o agrupar en un caso de uso que agrupe varias de estas operaciones cuando se realizan sobre unos mismos datos.
Sigla **CRUD**

disponibilidad *f* Capacidad de un sistema informático de estar disponible para sus usuarios. Si un usuario no puede acceder al sistema por el motivo que sea, decimos que el sistema

"no está disponible". Normalmente, la no disponibilidad se mide en porcentaje de tiempo (90%, 99%, etc.).

escenario *m* Secuencia de acciones e interacciones que se producen en un caso de uso bajo ciertas condiciones, expresadas sin (o con pocas) ramificaciones condicionales.

escenario principal *m* Escenario descrito completamente por un caso de uso desde el inicio del caso de uso hasta su compleción y que lleva al actor principal a cumplir su objetivo. A pesar de que es un escenario de éxito, no debe ser necesariamente el único del caso de uso.

extends Nombre oficial de la relación de extensión entre casos de uso en UML.
Ved **extensión** (2)

extensión (de un caso de uso) *f* 1. Un fragmento de escenario que empieza bajo cierta condición desde otro escenario. 2. Relación entre dos casos de uso en la que un caso de uso que denominamos *de extensión* describe una extensión de otro caso de uso que denominamos *de base*, de tal manera que el caso de uso de base no tiene ninguna referencia al caso de uso de extensión, sino que el caso de uso de extensión indica dónde del caso de uso de base se produce la extensión.

fiabilidad *f* Capacidad de un sistema informático de proporcionar información correcta de manera segura (es decir, tolerando posibles errores por parte de los usuarios o del hardware).

garantía mínima (de un caso de uso) *f* Aquello que podemos asegurar que sucederá al finalizar el caso de uso, sea cual sea el escenario en el que se produzca.

historia de usuario *f* Técnica de documentación de requisitos que pone énfasis en la comunicación verbal y minimiza la documentación escrita. Una historia de usuario está formada por tres componentes: el nombre, la conversación entre los desarrolladores y los clientes y la lista de pruebas que hay que hacer para verificar que se ha implementado correctamente.

include Nombre oficial de la relación de inclusión entre casos de uso en el lenguaje UML.
Ved **inclusión**

inclusión (entre casos de uso) *f* Relación entre dos casos de uso en la que en un paso de un escenario de un caso de uso se llama a otro caso de uso (que denominamos *subcaso de uso*).

mantenibilidad *f* Capacidad de un sistema informático de ser fácil de mantener. Es decir, que sea tan fácil corregir los errores que se encuentren (mantenimiento correctivo) como añadir nuevas funcionalidades (mantenimiento evolutivo).

pila de producto *f* Ved ***backlog***

portabilidad *f* Capacidad de un sistema informático de funcionar sobre más de una plataforma tecnológica o de funcionar sobre una plataforma diferente a aquella para la que fue desarrollado.

precondición *f* Condición que se ha de cumplir previamente. En el caso de los casos de uso, condiciones que se deben dar antes de poderse ejecutar el caso de uso.

product backlog Ved ***backlog***

prototipo *m* Producto que se crea para demostrar la funcionalidad del producto final. El prototipo sólo sirve para obtener información sobre cómo debe ser el producto final, pero no forma parte de él (es decir, una vez obtenida la información que se quería obtener el prototipo se descarta), por lo que ni siquiera es necesario usar la misma tecnología que se empleará para producir el producto final.

rendimiento *m* Medida de la relación entre el volumen de trabajo realizado y el consumo de recursos empleado.

representante (de un *stakeholder*) *m* Persona que actúa como intermediario entre el *stakeholder* y el equipo de desarrollo.

requerimiento *m* Acción o efecto de requerir. A menudo se emplea como sinónimo de requisito.

requisito *m* Condición exigida o necesaria para una cosa. En el campo de la ingeniería del software, necesidad o restricción que afecta a un producto de software y define qué soluciones son adecuadas (lo cumplen) y cuáles no (no lo cumplen) desde el punto de vista

de esta necesidad o restricción. Una solución será adecuada si satisface todos los requisitos del sistema.

requisito candidato *m* Necesidades o restricciones obtenidas en una primera etapa de obtención de requisitos que se convertirán en requisitos sólo si son seleccionadas como tales a la hora de definir el alcance del proyecto que se ha de desarrollar.

requisito funcional *m* Requisito que describe la funcionalidad esperada del sistema; es decir, que describe el comportamiento del sistema ante los estímulos que le llegan del exterior.
Ved también **requisito no funcional**

requisito no funcional *m* Requisito que restringe el conjunto de soluciones posible, normalmente a modo de restricción, sin hablar de la funcionalidad.

seguridad *f* Capacidad de un sistema informático de proteger la información ante robos, usos fraudulentos, corrupción de los datos, etc.

stakeholder *m* Persona o entidad con un interés sobre el producto que estamos desarrollando.

subcaso de uso *m* Caso de uso que es incluido en otro caso de uso.
Ved **inclusión**

triangulación *f* Técnica de valoración que consiste en obtener la estimación de un requisito a partir de dos requisitos previamente estimados (uno más complejo y otro más sencillo).

usabilidad *f* Capacidad de un sistema informático de ser fácil de utilizar.

usuario *m* Persona que utiliza un sistema informático.

Bibliografía

Bibliografía principal

Cockburn, A. (2001). *Writing Effective Use Cases*. Addison-Wesley.

Este libro nos da indicaciones sobre cómo se deben aplicar, de manera eficaz, los casos de uso en la gestión de requisitos. A pesar de centrarse en el uso de casos de uso para la documentación de requisitos, gran parte del discurso es aplicable con independencia del estilo de documentación.

Bibliografía complementaria

Davis, A. M. (2005). *Just Enough Requirements Management*. Dorset House.

En esta obra, Alan Davis propone un enfoque pragmático para la obtención y la gestión de requisitos.

Cohn, M. (2004). *User Stories Applied*. Addison Wesley.

Este libro trata, principalmente, de la técnica de las historias de usuario, pero –al igual que el de Cockburn– gran parte del discurso es aplicable a la obtención, gestión y documentación de requisitos con independencia del estilo de documentación.

Referencias bibliográficas

Volere Requirements Specification Template. http://www.volere.co.uk/template.htm (Última visita: septiembre del 2010).

Leffingwell, D.; Widrig, D. (2000). *Managing Software Requirements - A Unified Approach*. Addison Wesley.

Fitnesse. http://fitnesse.org/ (Última visita: septiembre del 2010).

Cucumber. http://cukes.info/ (Última visita: septiembre del 2010).

Autores varios (2010). *Unified Modeling Language™ (UML®)*. Ob-ject Management Group.

Alistair Cockburn. http://alistair.cockburn.us/use+case+questions (Última visita: septiembre del 2010).

Larry Constantine. *Users, Roles and Personas*.
http://www.foruse.com/articles/rolespersonas.pdf (Última visita: septiembre del 2010).