3.1. Presentación del capítulo

Este capítulo trata sobre entender los requisitos del sistema. Examinamos los detalles del workflow de requisitos de UP y presentamos la noción de requisitos. También presentamos una extensión de UP para tratar con requisitos sin utilizar casos de uso de UML.

3.2. El workflow de requisitos

Como se muestra en la figura 3.2, la mayor parte del trabajo en el workflow de requisitos ocurre en las fases de comienzo y elaboración justo al principio del ciclo de vida del proyecto. Esto no es sorprendente ya que no puede progresar más allá de la elaboración hasta que no sepa lo que va a crear.

Antes de que pueda incluso empezar a trabajar en el análisis y diseño orientado a objetos, tiene que tener alguna idea de lo que trata de conseguir, y ésta es la finalidad del workflow de requisitos. Desde el punto de vista del analista/diseñador orientado a objetos, la finalidad es descubrir y llegar a un acuerdo sobre lo que debería hacer el sistema, expresado en el idioma de los usuarios del sistema. Crear una especificación de alto nivel para lo que debería hacer el sistema es parte de la ingeniería de requisitos.

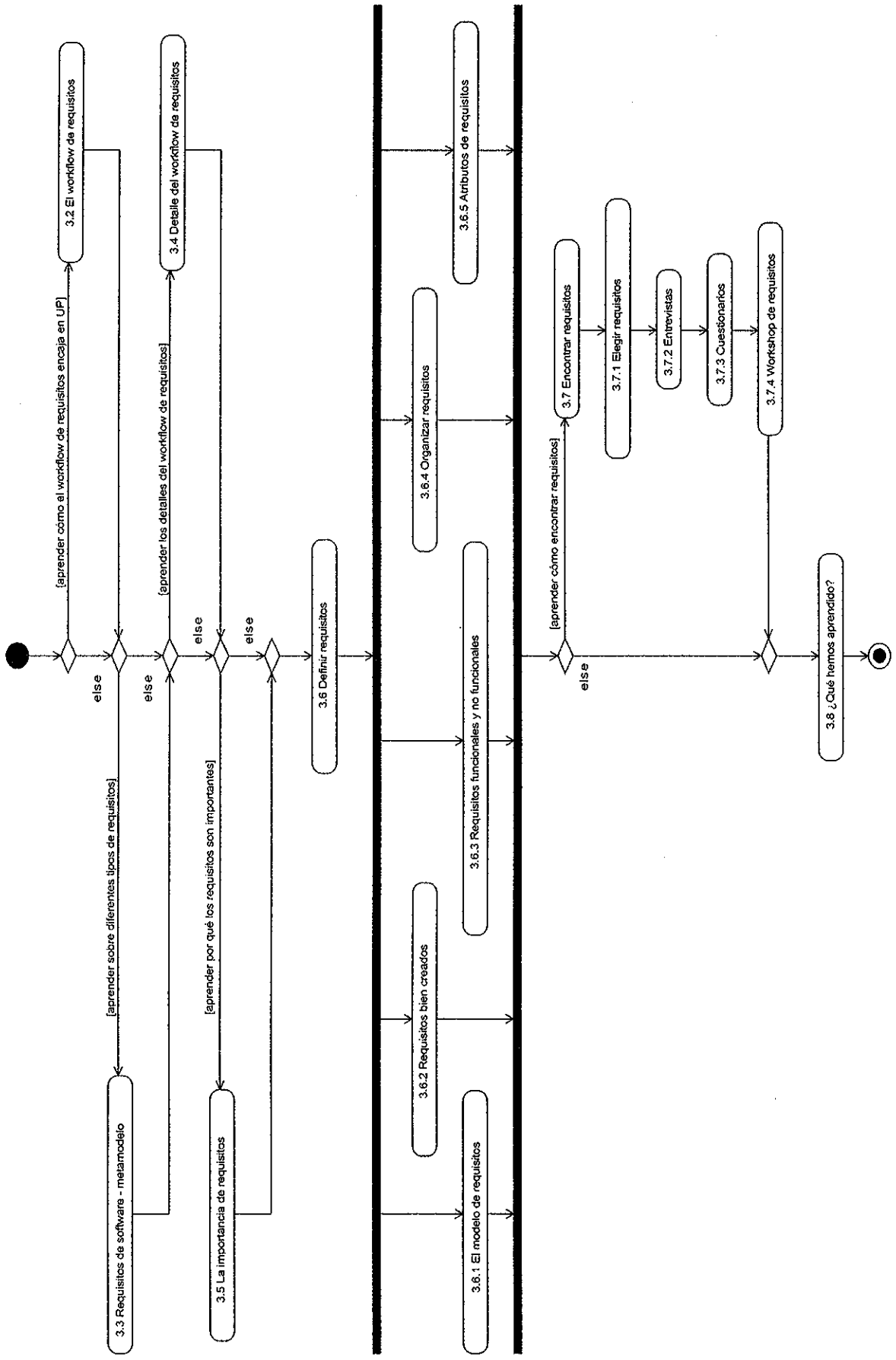

[aprender cómo el workflow de requisitos encaja en UP]
3.2 El workflow de requisitos
else
[aprender sobre diferentes tipos de requisitos]
3.3 Requisitos de software - metamodelo
else
[aprender los detalles del workflow de requisitos]
3.4 Detalle del workflow de requisitos
else
[aprender por qué los requisitos son importantes]
3.5 La importancia de requisitos
else
3.6 Definir requisitos
3.6.1 El modelo de requisitos
3.6.2 Requisitos bien creados
3.6.3 Requisitos funcionales y no funcionales
3.6.4 Organizar requisitos
3.6.5 Atributos de requisitos
[aprender cómo encontrar requisitos]
else
3.7 Encontrar requisitos
3.7.1 Elegir requisitos
3.7.2 Entrevistas
3.7.3 Cuestionarios
3.7.4 Workshop de requisitos
3.8 ¿Qué hemos aprendido?

Figura 3.1.

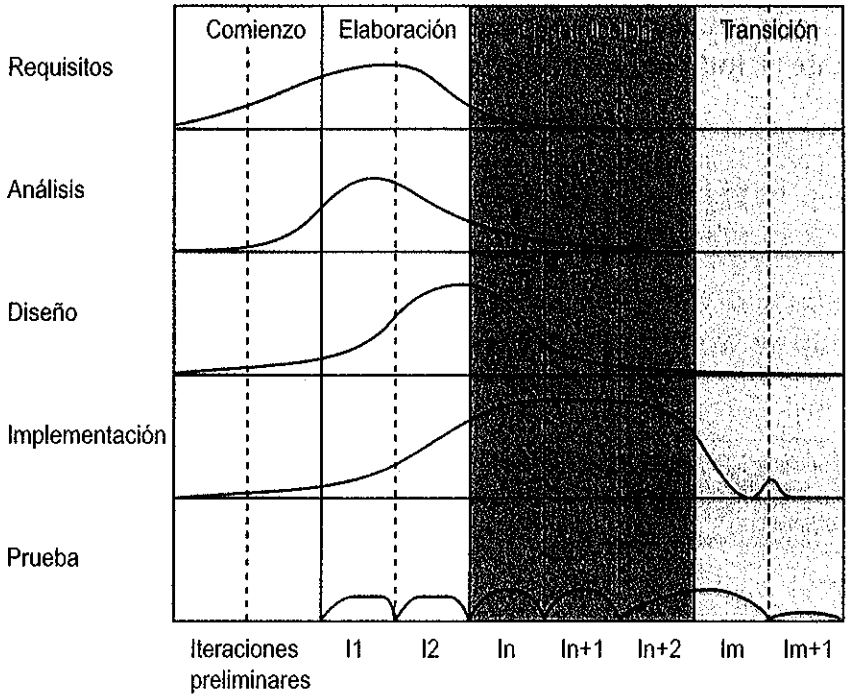


Figura 3.2. Adaptada de figura 1.5 [Jacobson 1] con permiso de Addison-Wesley.

Para cualquier sistema dado, existen muchos grupos de decisión: muchos tipos de usuario, ingenieros de mantenimiento, personal de soporte, agentes de venta, directores, etc. La ingeniería de requisitos va sobre obtener y priorizar los requisitos que estos grupos de decisión tienen para el sistema. Es un proceso de negociación ya que a menudo existen requerimientos en conflicto que se deben equilibrar. Por ejemplo, un grupo podría querer añadir muchos usuarios, lo que tendría como resultado un tráfico nada realista sobre la base de datos existente y la infraestructura de comunicaciones. Éste es un conflicto común en el momento ya que cada vez más empresas abren partes de sus sistemas a una amplia base de usuarios por medio de Internet.

Algunos libros de UML (y por lo tanto cursos de formación) indican que la noción de UML de los casos de uso es la única forma de capturar requisitos, pero esta afirmación no resiste al examen cuidadoso. Los casos de uso solamente pueden capturar requisitos funcionales, que son declaraciones sobre qué hará el sistema. Sin embargo, existe otro conjunto de requisitos no funcionales que son declaraciones sobre restricciones en el sistema (rendimiento, fiabilidad, etc), que no son apropiados para capturarse por los casos de uso. Por lo tanto, presentamos en este libro un enfoque robusto de ingeniería de requisitos por medio del cual ilustramos formas potentes y complementarias para capturar ambos conjuntos de requisitos.

3.3. Requisitos de software: metamodelo

La figura 3.3 muestra el metamodelo para nuestro enfoque de la ingeniería de requisitos en este libro. Contiene numerosa sintaxis UML que todavía no hemos

tratado. No se preocupe. Trataremos todo esto en detalle más adelante. Por ahora, lo siguiente es todo lo que necesita saber.

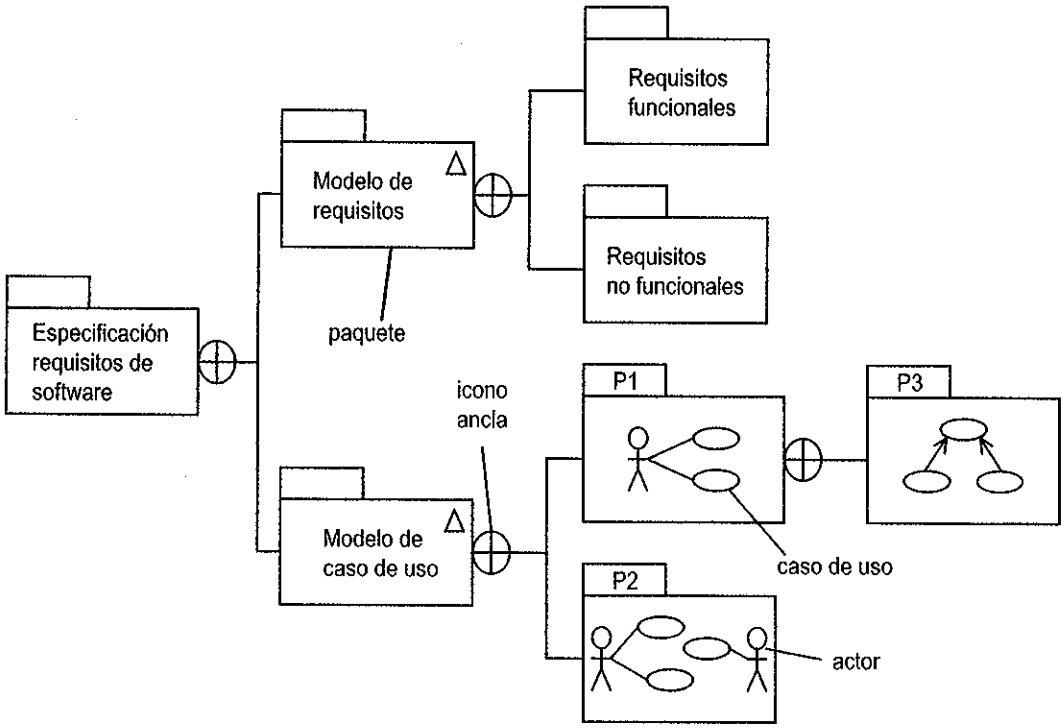


Figura 3.3.

- Los iconos que se parecen a carpetas son paquetes UML. Éstos son los mecanismos de agrupación de UML y contienen grupos de elementos de modelado de UML. En efecto, actúan de forma muy similar a las carpetas reales en un sistema de archivos en que se utilizan para organizar y agrupar elementos relacionados. Cuando el paquete tiene un pequeño triángulo en su esquina superior derecha, esto indica que el paquete contiene un modelo.
- El icono de ancla indica que el elemento en el extremo del círculo contiene el elemento en el otro extremo de la línea.

Nuestro metamodelo muestra que la especificación de requisitos de software (*Software requirements specification*, SRS) contiene un modelo de requisitos y un modelo de caso de uso. Estos dos modelos son formas diferentes, aunque complementarias, de capturar requisitos del sistema.

Puede ver que el modelo de requisitos contiene requisitos funcionales (requisitos que especifican lo que el sistema debería hacer) y requisitos no funcionales (requisitos que expresan restricciones no funcionales del sistema).

El modelo de caso de uso contiene muchos paquetes de caso de uso (solamente mostramos tres aquí) que contienen casos de uso (especificaciones de funcionalidad del sistema), actores (roles externos que interactúan directamente con el sistema) y relaciones.

La SRS es el principio del proceso de construcción del software. Es normalmente la entrada inicial hacia el análisis y diseño orientado a objetos.

Tratamos los requisitos en detalle en el resto de este capítulo, y los casos de uso y los actores en el siguiente.

3.4. Detalle del workflow de requisitos

La figura 3.4 muestra las tareas específicas para el workflow de requisitos de UP. Un diagrama como éste se conoce como un detalle de workflow ya que detalla las tareas de componentes de un workflow específico.

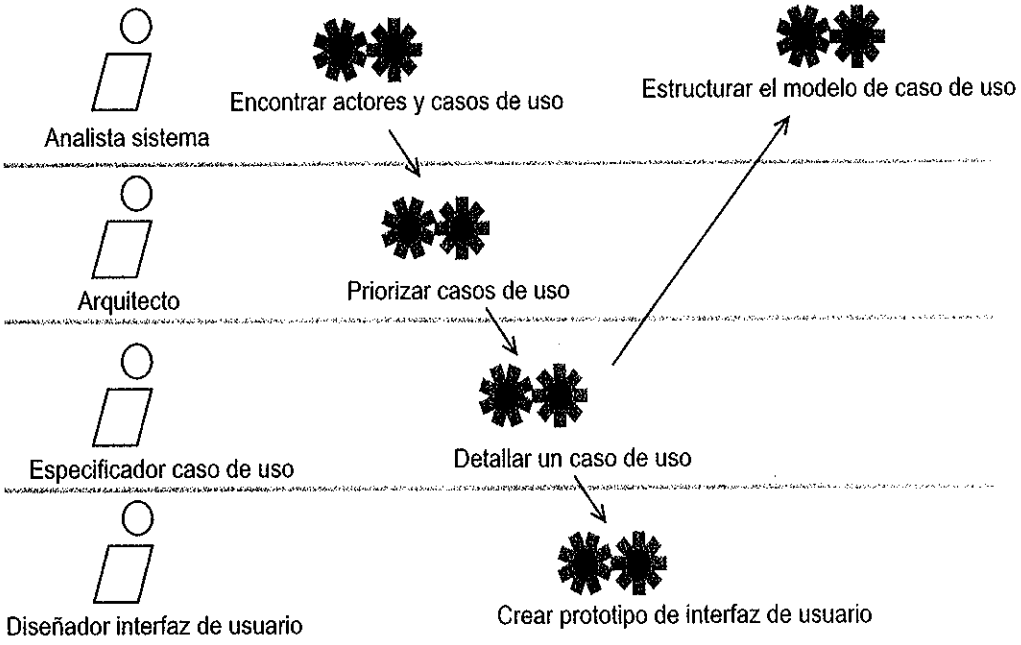


Figura 3.4. Reproducida de la figura 7.10 [Jacobson 1] con permiso de Addison-Wesley.

Los detalles de un workflow UP son modelados como recursos (los iconos a la izquierda) y actividades (los iconos que parecen ruedas dentadas). Las variantes UP como RUP pueden utilizar diferentes iconos, pero la semántica es la misma (véase el capítulo 2 para una breve explicación de las relaciones entre UP y RUP). Las flechas son relaciones que muestran el flujo normal de trabajo desde una tarea a la siguiente. Sin embargo, merece la pena tener en cuenta que esto es solamente una aproximación del workflow en el caso "promedio" y puede no ser una representación exacta particular de lo que pasa en la práctica. En el mundo real, puede esperar que algunas tareas se realicen en un orden diferente o en paralelo según las circunstancias. Puesto que éste es un libro de análisis y diseño, nos centramos solamente en las tareas importantes para los analistas y diseñadores orientados a objetos. En este caso, estamos interesados en las siguientes tareas:

- Encontrar actores y casos de uso.
- Detallar un caso de uso.
- Estructurar el modelo de caso de uso.

Las otras tareas en el workflow de requisitos no son relevantes para nosotros como analistas/diseñadores. Priorizar los casos de uso es una actividad de arquitectura y planificación de proyecto, y crear un prototipo de la interfaz de usuario es una actividad de programación. Si lo necesita, puede aprender más sobre estas actividades en [Jacobson 1].

En la figura 3.4 puede ver que el workflow estándar de UP se centra en casos de uso para la exclusión de cualquier otra técnica de obtención de requisitos. Eso es estupendo pero como hemos dicho, no aborda bien los aspectos no funcionales de los requisitos. Para trabajar con los requisitos de forma rigurosa, realizamos una sencilla extensión en el workflow de requisitos de UP para añadir nuevas tareas.

- Encontrar requisitos funcionales.
- Encontrar requisitos no funcionales.
- Priorizar requisitos.
- Hacer un seguimiento de requisitos para casos de uso.

También hemos incorporado un nuevo recurso, el ingeniero de requisitos. Las nuevas tareas y recursos se muestran en la figura 3.5.

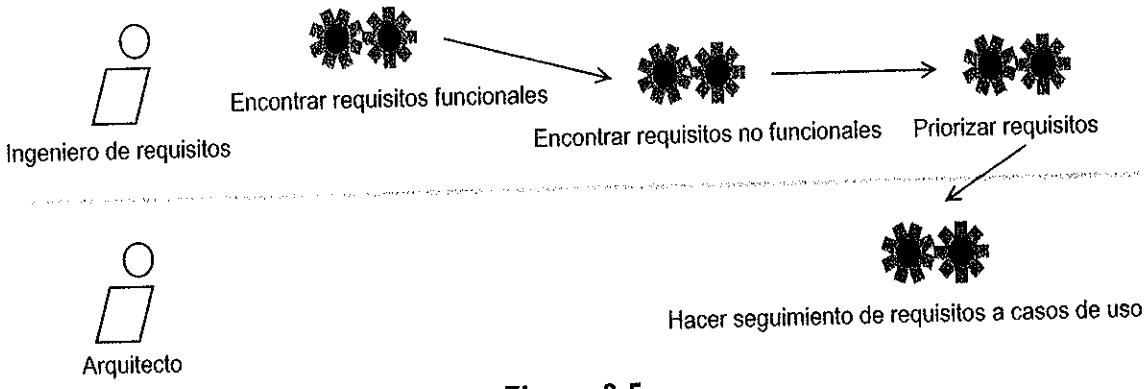


Figura 3.5.

3.5. La importancia de los requisitos

La ingeniería de requisitos es un término utilizado para describir las actividades implicadas en la obtención, documentación y mantenimiento de un conjunto de requisitos para un sistema de software. Trata acerca de descubrir qué necesitan los grupos de decisión que el sistema haga por ellos.

Según [Standish 1], requisitos incompletos y la ausencia de implicación del usuario eran las dos razones principales citadas para el fracaso del proyecto. Estos aspectos son fallos en la ingeniería de requisitos.

Puesto que el sistema de software final se basa sobre un conjunto de requisitos, la ingeniería de requisitos es un factor crítico de éxito en proyectos de desarrollo de software.

3.6. Definir requisitos

Podemos definir un requisito como "una especificación de lo que se debería implementar". Existen básicamente dos tipos de requisitos:

- **Requisitos funcionales:** Qué comportamiento debería ofrecer el sistema.
- **Requisitos no funcionales:** Una propiedad específica o restricción del sistema.

Los requisitos son (o al menos deberían ser) la base de todos los sistemas. Son básicamente una declaración de lo que debería hacer el sistema. En principio, los requisitos deberían ser solamente una declaración de lo que debería hacer el sistema, y no cómo lo debería hacer. Ésta es una importante distinción. Podemos especificar lo que un sistema debería hacer y qué comportamiento debería mostrar un sistema sin decir necesariamente nada sobre cómo esta funcionalidad debería estar realizada.

Aunque es realmente atractivo, en teoría, separar el "qué" del "cómo", en la práctica, un conjunto de requisitos tenderá a ser una mezcla de "qué" y "cómo". Esto es en parte porque a menudo es más sencillo escribir y entender una descripción de implementación en lugar de una declaración abstracta del problema, y en parte porque existen restricciones de implementación que predeterminan el "cómo" del sistema.

A pesar del hecho de que el comportamiento del sistema y, en último término, la satisfacción del usuario final se basa sobre la ingeniería de requisitos, muchas empresas no reconocen esto como una disciplina importante. Como hemos visto, la razón principal de que los proyectos de software fracasen se debe a problemas en requisitos.

3.6.1. El modelo de requisitos

Muchas empresas todavía no tienen una noción formal de requisitos o de un modelo de requisitos. El software se especifica en uno o más "documentos informales de requisitos" que a menudo se escriben en lenguaje natural y se presentan en cualquier forma y tamaño y en varios grados de utilidad. Para cualquier documento de requisitos, en cualquier forma, las preguntas clave son, "¿qué utilidad tiene para mí?" y "¿me ayuda a entender lo que el sistema debería hacer o no?". Desafortunadamente, muchos de estos documentos informales son solamente de utilidad limitada.

UP tiene un enfoque formal a los requisitos basándose en un modelo de caso de uso y lo ampliamos aquí con un modelo de requisitos basándose en ideas tradicionales de requisitos funcionales y no funcionales. Esta extensión está en relación directa al enfoque más sofisticado a la ingeniería de requisitos en RUP. Nuestro metamodelo de requisitos (véase la figura 3.3) muestra que la SRS consta de un modelo de caso de uso y un modelo de requisitos.

El modelo de caso de uso normalmente se crea en una herramienta de modelado UML como Rational Rose. Trataremos los casos de uso en detalle en el capitulo 4 y

5. El modelo de requisitos se puede crear en texto o en herramientas de ingeniería de requisitos especiales como RequisitePro (www.ibm.com) o DOORS (www.telelogic.com). Le recomendamos que utilice herramientas de ingeniería de requisitos si es posible. En los siguientes apartados examinamos cómo escribir requisitos bien formados.

3.6.2. Requisitos bien formados

UML no proporciona ninguna recomendación sobre escribir requisitos tradicionales. De hecho, UML trata con requisitos por medio del mecanismo de casos de uso que examinamos más adelante. Sin embargo, muchos modeladores (nosotros incluidos) creen que los casos de uso no son suficientes y que seguimos necesitando requisitos tradicionales y herramientas de administración de requisitos.

Recomendamos un formato muy sencillo para indicar requisitos (véase la figura 3.6). Cada requisito tiene un identificador único (normalmente un número), una palabra clave (debería) y una declaración de función. La ventaja de adoptar una estructura uniforme es que herramientas de administración de requisitos como DOORS pueden analizar los requisitos más fácilmente.

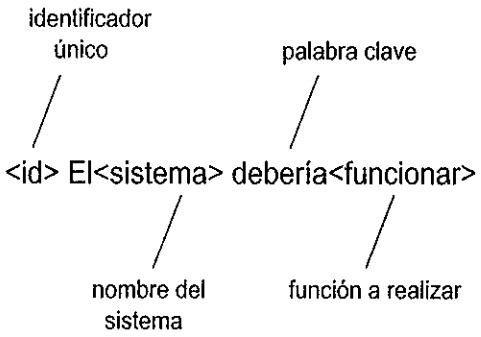


Figura 3.6.

3.6.3. Requisitos funcionales y no funcionales

Es de utilidad dividir los requisitos en requisitos funcionales y no funcionales. Existen muchas otras formas de categorizar los requisitos, pero mantendremos las cosas lo más sencillas posible y trabajaremos inicialmente con estas dos categorías.

Un requisito funcional es una declaración de lo que debería hacer el sistema; es una declaración de la función del sistema. Por ejemplo, si recopila requisitos para un cajero automático, podría identificar los siguientes requisitos funcionales:

1 El cajero debería comprobar la validez de la tarjeta insertada.

2. El cajero debería validar el número PIN facilitado por el cliente.

3. El cajero debería dispensar no más de 250 euros para cualquier tarjeta de cajero electrónico en cualquier período de 24 horas.

Un requisito no funcional es una restricción situada sobre el sistema. Para su cajero automático puede haber los siguientes requisitos no funcionales.

1. El sistema del cajero se debería escribir en C++.
2. El sistema del cajero debería comunicarse con el banco al utilizar un cifrado de 256 bits.
3. El sistema del cajero debería validar una tarjeta de cajero electrónico en tres segundos o menos.
4. El sistema del cajero debería validar un PIN en tres segundos o menos.

Puede ver que los requisitos no funcionales especifican, o restringen, cómo se debería implementar el sistema.

3.6.4. Organizar requisitos

Si utiliza una herramienta de administración de requisitos, podrá organizar sus requisitos en una taxonomía. Ésta es una jerarquía de tipos de requisitos que puede utilizar para categorizar sus requisitos. La razón principal para utilizar tipos de requisitos es que pueden organizar un amplio conjunto de requisitos sin estructurar en dominios más pequeños y más manejables. Esto debería ayudarle a trabajar con los requisitos de forma más eficaz.

La división básica en requisitos funcionales y no funcionales que hemos descrito anteriormente es una taxonomía muy sencilla, pero podría por ejemplo categorizar aún más sus requisitos al extender esta taxonomía como se muestra en la figura 3.7.

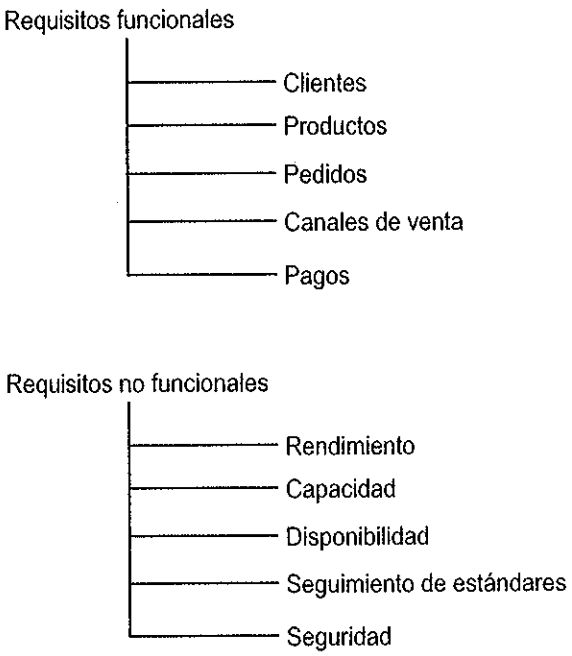


Figura 3.7.

Los tipos específicos de requisito que elija dependen del tipo de software que construya. Esto es especialmente cierto para requisitos funcionales. Para requisitos no funcionales, el conjunto de tipos de requisitos mostrado en la figura 3.7 es bastante estándar y proporciona un buen punto de partida.

Organizar requisitos por tipo es un enfoque de utilidad si tiene que tratar con muchos requisitos (más de un centenar). Es especialmente de utilidad si utiliza una herramienta de ingeniería de requisitos ya que esto le permitirá consultar el modelo de requisitos por tipo de requisito para extraer información de utilidad.

En principio, su jerarquía de tipos de requisito puede ser tan profunda como guste. En la práctica, dos o tres niveles parecen ser correctos a menos que trabaje en un sistema muy complejo.

3.6.5 Atributos de requisitos

Todo requisito puede tener un conjunto de atributos que captura información adicional (metadatos) sobre el requisito.

Todo atributo de requisito tiene un nombre descriptivo y un valor. Por ejemplo, un requisito puede tener un atributo denominado `fechaVencimiento` que tiene como su valor la fecha en la que se tiene que entregar el requisito. El requisito podría tener también un atributo `origen` que tiene como su valor una descripción de dónde se ha originado el requisito. El conjunto preciso de atributos que decida utilizar depende de la naturaleza y necesidades de su proyecto y puede variar por tipo de requisito.

Quizás el atributo de requisito más común es `prioridad`. El valor de este atributo es la prioridad del requisito relativa a todos los otros requisitos. Un esquema común para asignar prioridad es el conjunto de criterios descrito en la tabla 3.1.

Tabla 3.1.

Valores de atributo de prioridad	Semántica
Debe tener.	Requisitos obligatorios que son fundamentales para el sistema.
Debería tener.	Requisitos importantes que se pueden omitir.
Podría tener.	Requisitos que son opcionales (se realizan si hay tiempo).
Quiere tener.	Requisitos que pueden esperar para versiones posteriores del sistema.

Cuando se utilizan estos criterios, cada requisito tiene un atributo `Prioridad` que puede adoptar uno de los valores. Las herramientas de ingeniería de requisitos generalmente le permiten consultar el modelo de requisitos por valor de atributo

para que pueda, por ejemplo, generar una lista de todos los requisitos de prioridad superior. Esto es de mucha utilidad.

La virtud de estos criterios es su simplicidad. Sin embargo, confunde dos atributos diferentes de un requisito: su importancia y su precedencia. La importancia de un requisito, una vez establecido, tiende a permanecer relativamente establece. Sin embargo, la precedencia de un requisito, cuando se realizará con respecto a otros requisitos, puede cambiar durante el curso del proyecto por razones no relacionadas con la importancia. Por ejemplo, la disponibilidad de recursos o dependencias sobre otros requisitos. RUP define un conjunto más completo de atributos de requisito que separan importancia (Beneficio) y precedencia (VersiónObjetivo). Los atributos RUP se resumen en la tabla 3.2.

Tabla 3.2.

Atributo	Semántica
Estado.	Esto puede tener uno de los siguientes valores: • **Propuesto:** Los requisitos siguen estando bajo deliberación y todavía no se han acordado. • **Aprobado:** Los requisitos que se han aprobado para implementación. • **Rechazado:** Los requisitos se han rechazado para implementación. • **Incorporado:** Los requisitos se han implementado para una versión en particular.
Beneficio.	Esto puede tener uno de los siguientes valores: • **Crítico:** El requisito se debe implementar; de lo contrario el sistema no será adecuado para los grupos de decisión. • **Importante:** El requisito se podría omitir pero esto afectaría a la usabilidad del sistema y la satisfacción de los grupos de decisión. • **Útil:** El requisito se podría omitir sin impacto importante en la aceptación del sistema.
Esfuerzo.	Una estimación del tiempo y los recursos necesarios para implementar la característica medida en personas/día o cualquier otra unidad (`www.ifpug.org`).
Riesgo.	El riesgo que implica añadir esta característica. Alto, Medio o Bajo.
Estabilidad.	Una estimación de la probabilidad de que el requisito cambiará de alguna forma. Alto, Medio o Bajo.
VersiónObjetivo.	La versión del producto en la que el requisito se debería implementar.

Tanto si utiliza los criterios anteriormente mencionados, RUP, u otro conjunto de atributos de requisitos, depende de su proyecto en particular. El punto clave cuando se define un conjunto de atributos es mantenerlo lo más sencillo posible. Elija solamente aquellos atributos que proporcionan beneficio a su proyecto. Si un atributo no proporciona beneficio, no lo utilice.

3.7. Encontrar requisitos

Los requisitos provienen del contexto del sistema que trata de modelar. Este contexto incluye:

- Usuarios directos del sistema.
- Otros grupos de decisión (por ejemplo, directores, mantenimiento, instaladores).
- Otros sistemas con el que interactúa el sistema.
- Dispositivos de hardware con el que interactúa el sistema.
- Restricciones legales y regulatorias.
- Restricciones técnicas.
- Objetivos de negocio.

La ingeniería de requisitos normalmente empieza con un documento de visión que detalla lo que va a hacer el sistema y los beneficios que proporcionará a un conjunto de grupos de decisión. La idea de este documento es captar los objetivos esenciales del sistema desde el punto de vista de los grupos de decisión. El documento de visión lo generan los analistas del sistema durante la fase de inicio del UP.

Después del documento de visión, empieza la ingeniería de requisitos. Examinaremos algunas técnicas para la creación de requisitos en los siguientes apartados.

3.7.1. Obtención de requisitos: el mapa no es el territorio

Siempre que trabaje con personas para definir los requisitos para un sistema de software, está tratando de obtener de ellos una imagen precisa, o mapa, de su modelo del mundo. Según Noam Chomsky, en su libro *Syntactic Structures* de 1975 [Chomsky 1] sobre gramática transformacional, el mapa se crea por los tres procesos de eliminación, distorsión y generalización. Esto es totalmente necesario puesto que no tenemos el equipamiento cognitivo para captar todos los detalles del mundo en un mapa mental infinitamente detallado, por lo que tenemos que ser selectivos. Realizamos nuestra selección de un amplio conjunto de información posible al aplicar estos tres filtros:

- **Eliminación:** La información se filtra.
- **Distorsión:** La información se modifica por los mecanismos relacionados de creación e ilusión.
- **Generalización:** La información se resume en reglas, creencias y principios sobre lo verdadero y lo falso.

Estos filtros moldean el lenguaje natural. Es importante conocerlos cuando está llevando a cabo una captura y análisis de requisitos detallados ya que puede necesitar identificarlos para recuperar información.

Aquí se presentan algunos ejemplos de un sistema de administración de biblioteca. Para cada uno existe una pregunta para el filtro y una posible respuesta.

- Ejemplo: "utilizan el sistema para tomar prestados libros". Eliminación
  - Pregunta: ¿Quién utiliza específicamente el sistema para tomar prestado libros?
  - Respuesta: Miembros de la biblioteca, otras bibliotecas y bibliotecarios.
- Ejemplo: "Los usuarios no pueden tomar prestado otro libro hasta que se hayan devuelto todos los libros para los que se ha acabado el plazo de préstamo". Distorsión.
  - Pregunta: ¿Existe alguna circunstancia bajo la que alguien podría tomar prestado un nuevo libro antes de que se hayan devuelto todos los libros a los que se les ha acabado el plazo de préstamo?
  - Respuesta: En realidad existen dos circunstancias bajo las cuales el derecho de un usuario a tomar prestado un libro se puede restaurar. En primer lugar, se devuelven todos los libros a los que se les ha acabado el plazo de préstamo; y en segundo lugar, se tiene que pagar por cualquier libro al que se le ha acabado el plazo de préstamo y no se haya devuelto.
- Ejemplo: "Todo el mundo debe tener una tarjeta para tomar prestados libros". Generalización.
  - Pregunta: ¿Existe algún usuario del sistema que podría no necesitar tener una tarjeta?
  - Respuesta: Algunos usuarios del sistema, como otras bibliotecas, pueden no necesitar una tarjeta o pueden tener un tipo especial de tarjeta con diferentes condiciones.

Los dos últimos casos son particularmente interesantes como ejemplos de un patrón común de lenguaje: el cuantificador universal. Los cuantificadores universales son palabras como:

- todos
- todo el mundo

- siempre
- nunca
- nadie
- ninguno

Siempre que encuentre un cuantificador universal, puede haber encontrado una eliminación, distorsión o generalización. A menudo indican que ha alcanzado los límites del mapa mental de alguien. Como tal, cuando se realiza análisis a menudo es una buena idea preguntar con cuantificadores universales. Casi escribimos "siempre es una buena idea preguntar con cuantificadores universales", pero en este caso, nos preguntamos a nosotros mismos.

3.7.2. Entrevistas

Entrevistar a los grupos de decisión es la forma más directa de recopilar requisitos. Normalmente es mucho mejor tener entrevistas uno a uno siempre que sea posible. Los puntos esenciales se anotan a continuación:

- **No invente una solución:** Puede pensar que tiene una muy buena idea de lo que necesitan los grupos de decisión, pero debe mantener esta preconcepción a un lado durante la entrevista. Ésta es la única forma de averiguar lo que realmente necesita.
- **Formule preguntas abiertas:** Éstas son preguntas que no presuponen ninguna respuesta particular y animan al entrevistado a hablar sobre el problema. Por ejemplo, "¿quién utiliza el sistema?" anima a hablar, mientras que ¿utiliza usted el sistema? Implica una respuesta sí/no y termina la discusión.
- **Escuche:** Ésta es la única forma en la que averiguará qué quieren los grupos de decisión, por lo tanto déjeles tiempo para hablar. Permítales hablar sobre una pregunta y que la exploren de su propia forma. Si busca respuestas específicas a preguntas, es posible que invente una solución y formule preguntas cerradas basándose en esa invención.
- **No adivine los pensamientos:** De hecho, todos lo hacemos hasta cierto punto. Adivinar los pensamientos es "pensar" que conoce lo que otra persona siente, quiere o piensa basándose en lo que usted sentiría, querría o pensaría en una situación similar. Ésta es una habilidad humana muy importante ya que es la base de la empatía. Sin embargo, puede cruzarse en su camino cuando trata de obtener requisitos y puede acabar con lo que usted necesita en lugar de con lo que necesitan los grupos de decisión.
- **Tenga paciencia.**

El contexto de la entrevista puede tener un gran impacto sobre la calidad de la información que recibe. Personalmente, preferimos contextos informales como una

cafetería, ya que permiten tanto al entrevistador como al entrevistado relajarse y hablar abiertamente. La mejor forma de captar información durante una entrevista es papel y lápiz. Escribir algo en un portátil distrae a ambas partes y puede ser bastante intimidador para el entrevistado. Nos gusta utilizar mapas mentales como una forma flexible, no amenazante y gráficamente valiosa de recopilar información. Puede averiguar más sobre esto en `www.mind-map.com`.

Después de una entrevista, analice la información y construya algunos requisitos candidatos.

3.7.3. Cuestionarios

Los cuestionarios no son sustitutos de las entrevistas. Si decide utilizar cuestionarios sin realizar ninguna entrevista, puede encontrarse en una situación imposible en la que debe decidir sobre una lista de preguntas antes de saber cuál es la pregunta correcta a formular.

Los cuestionarios pueden ser un suplemento de utilidad para las entrevistas. Son excelentes para conseguir respuestas para preguntas específicas cerradas. Puede descubrir preguntas claves a partir de las entrevistas e incorporar éstas en un cuestionario que puede distribuir a una audiencia más amplia. Esto le puede ayudar a validar su entendimiento de los requisitos.

3.7.4. Workshop de requisitos

Ésta es una de las formas más eficaces de capturar requisitos. Consigue que los grupos de decisión clave participen en un workshop para identificar requisitos clave. El workshop debería centrarse en la tormenta de ideas. Ésta es una técnica potente para captar información. Los participantes en la reunión deberían ser un facilitador, un ingeniero de requisitos y los grupos de decisión clave así como expertos. El procedimiento es el siguiente:

1. Explicar que esto es una tormenta de ideas.
    1. Todas las ideas se aceptan como buenas ideas.
    2. Las ideas se graban pero no se debaten; nunca se discute sobre algo, simplemente se escribe y se sigue adelante. Todo se analizará más adelante.
2. Pida a los miembros del equipo que nombren los requisitos clave para el sistema.
    1. Escribir cada requisito en una tarjeta.
    2. Pegar la tarjeta en la pared o en la pizarra.
3. Luego, puede decidir pasar por los requisitos identificados y anotar atributos adicionales en cada uno de ellos, como se ha visto anteriormente.

Después de la reunión, analice los resultados y conviértalos en requisitos como hemos tratado anteriormente en este capítulo. Haga circular los resultados para comentarios. La ingeniería de requisitos es un proceso iterativo en el que pone al descubierto los requisitos a medida que mejora su conocimiento de las necesidades de los grupos de decisión. Es posible que necesite mantener varios workshops con el tiempo a medida que amplía su conocimiento.

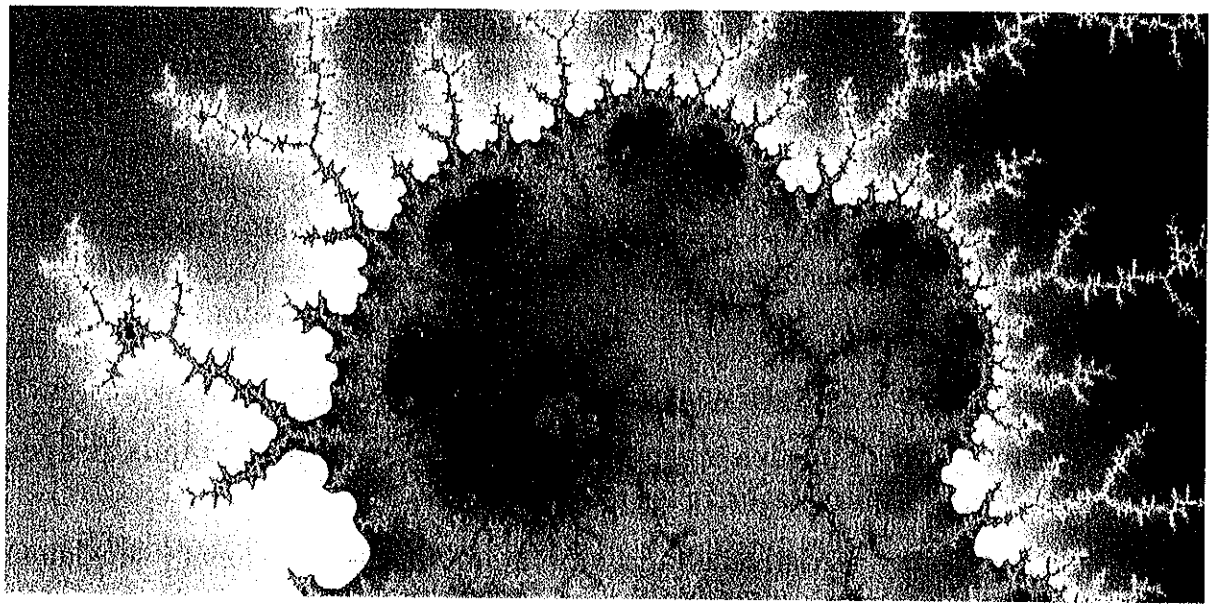
Puede encontrar más información sobre la realización de workshops y la ingeniería de requisitos en general en [Leffingwell 1] y [Alexander 1].

3.8. ¿Qué hemos aprendido?

Este capítulo ha presentado el workflow de requisitos de UP y una explicación general de los requisitos de software. Ha aprendido lo siguiente.

- La mayor parte del trabajo en el workflow de requisitos ocurre en las fases del comienzo y elaboración del ciclo de vida del proyecto de UP.
- Nuestro metamodelo de requisitos (figura 3.3) muestra que existen dos formas de capturar requisitos, como requisitos funcionales y no funcionales y como casos de uso y actores.
- El detalle del workflow de requisitos de UP contiene las siguientes actividades que son de interés para nosotros como analistas y diseñadores orientados a objetos:
  - Encontrar actores y caso de uso.
  - Detallar un caso de uso.
  - Estructurar el modelo de caso de uso.
- Ampliamos el workflow estándar de requisitos de UP con:
  - Actor: Ingeniero de requisitos.
  - Actividad: Encontrar requisitos funcionales.
  - Actividad: Encontrar requisitos no funcionales.
  - Actividad: Priorizar requisitos.
  - Actividad: Hacer un seguimiento de los requisitos para casos de uso.
- La mayor parte de los fracasos de proyectos se deben a problemas con la ingeniería de requisitos.
- Existen dos tipos de requisitos:
  - Requisitos funcionales: Qué comportamiento debería ofrecer el sistema.
  - Requisitos no funcionales: Una propiedad o restricción específica en el sistema.

- Los requisitos bien formulados se deberían expresar en lenguaje sencillo con declaraciones `debería`, de modo que se puedan analizar fácilmente por herramientas de ingeniería de requisitos.
  - `<id>`El `<sistema>` debería `<funcionar>`.
- El modelo de requisitos contiene los requisitos funcionales y no funcionales para un sistema. Esto puede ser:
  - Un documento.
  - Una base de datos en una herramienta de administración de requisitos.
- Los requisitos se pueden organizar en una taxonomía, una jerarquía de tipos de requisitos que categoriza los requisitos.
- Los requisitos pueden tener atributos, la información adicional (metadatos) asociada con cada requisito:
  - Por ejemplo, prioridad: debe tener, debería tener, podría tener y quiere tener.
  - Por ejemplo, atributos RUP (Estado, Beneficio, Esfuerzo, Riesgo, Estabilidad y VersiónObjetivo).
  - Mantener los requisitos de atributo en el mínimo que beneficie a su proyecto.
- El mapa no es el territorio. El lenguaje natural contiene:
  - Eliminaciones: La información se filtra.
  - Distorsiones: La información se modifica.
  - Generalizaciones: La información se resumen en reglas, creencias y principios sobre lo verdadero y falso.
- Los cuantificadores universales (por ejemplo "todos", "siempre") pueden indicar la frontera del mapa mental del mundo de alguien. Debería preguntarles.
- Técnicas para encontrar requisitos:
  - Entrevistas.
  - Cuestionarios.
  - Workshops.



4

Modelado del caso de uso