Educción de Requisitos

Un error común cometido por la gente cuando se trata de diseñar algo a prueba de tontos es subestimar la ingenuidad de los realmente tontos. —Douglas Adams, en *Mostly Harmless*

Un requisito es una característica que el sistema debe tener, o una restricción a la cual satisfacer para ser aceptado por el cliente. La ingeniería de requisitos se encarga de definir los requisitos del sistema en construcción, e incluye dos actividades principales; educción de requisitos, que resulta en la especificación del sistema que entiende el cliente, y el análisis, que resulta en un modelo de análisis que los desarrolladores pueden interpretar sin ambigüedad. La educción de requisitos es la más desafiante de las dos, pues requiere de la colaboración de varios grupos de participantes con diferentes habilidades. Por un lado, el cliente y el usuario son expertos en su dominio y tienen una idea general de qué debe hacer el sistema, pero por lo general tienen poca experiencia en el desarrollo de software. Por otro lado, los desarrolladores tienen experiencia en la construcción de sistemas, pero, por lo general, poco conocimiento sobre el entorno diario de los usuarios.

Los escenarios y casos de uso proveen herramientas para cerrar esta brecha. Un *escenario* describe un ejemplo de uso del sistema en base a una serie de interacciones entre el usuario y éste. Un *caso de uso* es una abstracción que describe una clase de escenario. Ambos, escenarios y casos de uso, se escriben en lenguaje natural, de forma entendible para el usuario.

En este capítulo, nos concentraremos en la educción de requisitos basada en escenarios. Los desarrolladores observando y entrevistando usuarios. Primero, los desarrolladores representan los procesos actuales de trabajo de los usuarios en escenarios tal y como son, después desarrollan escenarios visionarios que describen la funcionalidad que proveerá el sistema. El cliente y los usuarios validan la descripción del sistema, revisando los escenarios y probando prototipos pequeños que proveen los desarrolladores. Según se vaya estabilizando y madurando la definición del sistema, los desarrolladores y el cliente acuerdan una especificación de requisitos en forma de requisitos funcionales, no funcionales, casos de uso y escenarios.

Introducción: Ejemplos de Usabilidad.

¿Pies o millas?

Durante un experimento con láser, un rayo láser fue disparado a un espejo en el Trasbordador Espacial Discovery. Se suponía que el láser sería reflejado hacia la cima de una montaña. El usuario introdujo la elevación de la montaña como “10,023”, suponiendo que las unidades de entrada eran en pies. La computadora interpretó el número en millas y el rayo láser fue reflejado lejos de la tierra, hacia una montaña hipotética de 10,023 millas de altura.

Punto decimal contra separador de millares

En Estados Unidos, los puntos decimales se representan con un punto (“.”) y los separadores de millares con una coma (“,”); en Alemania es al revés. Suponga que un usuario en Alemania, consciente de ambas convenciones, está viendo un catálogo con precios en dólares ¿Qué convención debería usarse?

Patrones Estándar

En el editor de texto Emacs, el comando <Control-x><Control-c> existe en el programa. Si se necesita guardar algún archivo, el editor le preguntará al usuario “¿Guardar archivo miDocumento.txt? (y o n)”. Si se responde y, el editor guardará el archivo antes de salir. Muchos usuarios se basan en este patrón y teclean de forma sistemática la secuencia <Control-x> <Control-c> seguido por una "y" al salir de un editor. Sin embargo, otros editores preguntan “¿Está seguro que desea salir? (y o n)”. Cuando los usuarios cambian de Emacs a un editor como estos, no podrán guardar su trabajo, a menos que se desacostumbren.

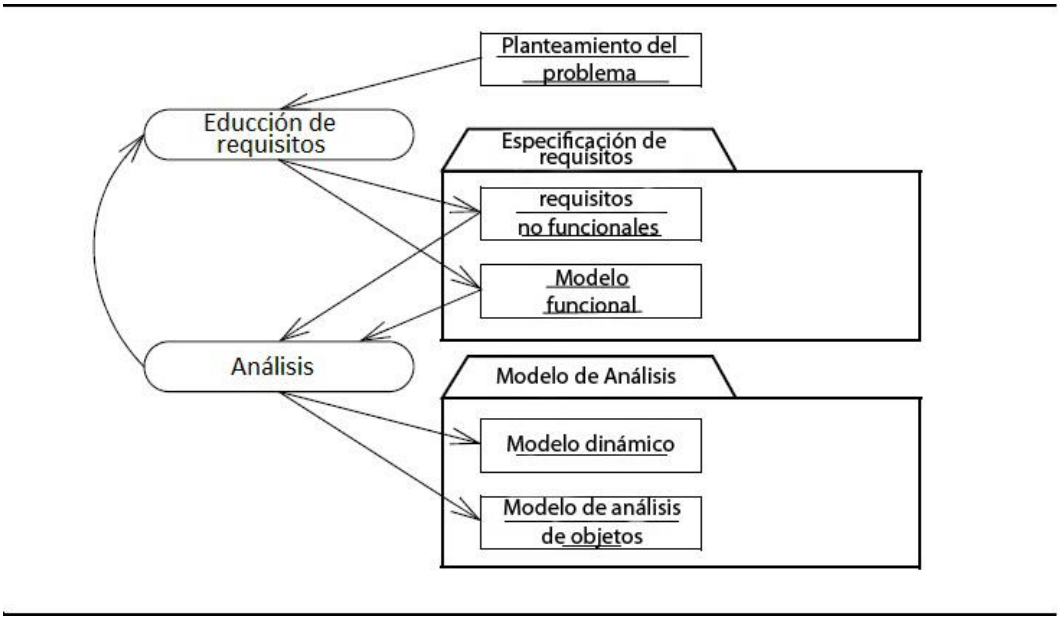
La educción de requisitos se trata de comunicación entre desarrolladores, clientes y usuarios para definir un sistema nuevo. El fallo en la comunicación se traduce en un sistema difícil de usar o que simplemente no ayuda al usuario con su trabajo. Los errores cometidos en la educción de requisitos son costosos de corregir, pues generalmente se descubren tarde en el proceso, a veces durante la entrega. Estos errores incluyen funcionalidades no brindadas al sistema, que se suponía tuviera, funcionalidades mal especificadas, interfaces de usuario engañosas o inutilizables, y funcionalidades obsoletas. Los métodos de educción de requisitos tienen como objetivo mejorar la comunicación entre desarrolladores, clientes y usuarios. Los desarrolladores construyen un modelo del dominio de la aplicación observando a los usuarios en su entorno. Seleccionan una representación que sea entendible por los clientes y los usuarios. Validan el modelo de dominio construyendo prototipos simples de la interfaz de usuario y recibiendo retroalimentación de usuarios potenciales. Un ejemplo de un prototipo simple es el mostrar la interfaz de usuarion sus elementos y botones. El usuario potencial puede manipular el menú de elementos y botones para saber cómo usar el sistema, pero sin

tener una respuesta real después de hacer clic sobre un botón, pues no se ha implementado la funcionalidad requerida.

Visión General de la Educción de Requisitos

La **educción de requisitos** se enfoca en la definición del propósito del sistema. El cliente, los desarrolladores y usuarios identifican un área problemática y definen un sistema que se encarga del problema. A esta definición se le llama **especificación de requisitos** y sirve como contrato entre el cliente y los desarrolladores. La especificación de requisitos se estructura y formaliza durante el análisis para producir un **modelo de análisis** (Figura 4-1). Ambos, la especificación de análisis y el modelo de análisis representan la misma información. Difieren solo en el lenguaje y notación que utilizan; la especificación se escribe en lenguaje natural, mientras que el modelo de análisis se expresa generalmente en notación formal o semi-formal. La especificación apoya la comunicación con el cliente y los usuarios. El modelo de análisis apoya la comunicación entre desarrolladores. Ambos son modelos del sistema en el sentido de que son un intento de representar acertadamente los aspectos externos del sistema. Sabiendo que ambos modelos representa los mismos aspectos del sistema, la educción y el análisis ocurren concurrentemente e iterativamente.

La educción de requisitos y el análisis se concentran solo en la percepción del usuario sobre el sistema. Por ejemplo, la funcionalidad del sistema, la interacción entre este y el usuario, los errores que el sistema puede detectar y manejar, y las condiciones del entorno en las que el sistema funciona son parte de los requisitos. La estructura del sistema, la selección de la tecnología implementada para construirlo, su diseño, la metodología de desarrollo, y otros aspectos que no son visibles directamente para el usuario no son parte de los requisitos.



La educción de requisitos incluye las siguientes actividades:

- **Identificación de Actores:** Los desarrolladores identifican los diferentes tipos de usuario a los que apoyará el sistema a crear.
- **Identificación de Escenarios:** Los desarrolladores observan a los usuarios y desarrollan un conjunto de escenarios detallados para la funcionalidad típica proveída por el sistema a crear. Los desarrolladores usan los escenarios para comunicarse con el usuario y profundizar su entendimiento del dominio de la aplicación
- **Identificación de Casos de Uso:** Una vez que los desarrolladores acuerdan un conjunto de escenarios, derivan de los escenarios un grupo de casos de uso que representan completamente el sistema a construir. Mientras que los escenarios son ejemplos concretos que ilustran un caso específico, los casos de uso son abstracciones que describen todos los casos posibles. Cuando se describe un caso de uso, los desarrolladores determinan el alcance del sistema.
- **Refinación de Casos de Uso:** Los desarrolladores se aseguran de que la especificación de requisitos esté completa detallando cada caso de uso y describiendo el comportamiento del sistema ante la presencia de errores y condiciones excepcionales.
- **Identificación de Relaciones entre Casos de Uso**: Los desarrolladores identifican las dependencias entre casos de uso. También consolidan el modelo de casos de uso factorizando la funcionalidad general. Esto asegura que la especificación de requisitos sea consistente.
- **Identificación de Requisitos No Funcionales**: Los desarrolladores, usuarios y clientes acuerdan los aspectos visibles para el usuario, pero que no estén

relacionados directamente con la funcionalidad. Esto incluye restricciones en la ejecución del sistema, su documentación, los recursos que consume, seguridad y calidad de este.

Durante la educción de requisitos, los desarrolladores acceden a diversas fuentes de información, incluyendo documentos provistos por el cliente acerca del dominio de la aplicación, manuales y documentación técnica de sistemas actuales/antiguos que el sistema a construir reemplazará, y lo más importante, los propios clientes y usuarios. Los desarrolladores interactúan mucho con usuarios y clientes durante la educción. Nos concentraremos en dos métodos de educción de información, tomar decisiones con usuarios y clientes, y el manejo de dependencias entre requisitos y otros artefactos:

- **Diseño de Aplicaciones en Conjunto (JAD):** se concentra en la construcción de consensos entre desarrolladores, usuarios, y clientes, desarrollando conjuntamente la especificación de requisitos.
- **Trazabilidad:** se concentra en el registro, estructuración, enlace, agrupación y mantenimiento de dependencias entre los requisitos y otros productos del trabajo.

Conceptos de la Educción de Requisitos

En esta sección, describimos los conceptos principales de la educción de requisitos.

Requisitos Funcionales

Los requisitos funcionales describen la interacción entre el sistema y su entorno independiente a su implementación. El entorno incluye al usuario y cualquier otro sistema externo con el que interactúe el sistema. Por ejemplo, la figura 4-2 muestra un requisito funcional para "SatWatch", un reloj que se resetea sin la intervención de un usuario:

SatWatch es un reloj de muñeca que muestra el tiempo basado en su locación actual. Usa satélites GPS para determinar su locación y estructuras internas de datos para convertir la locación en una zona horaria.

La información almacenada en *SatWatch* y su precisión en la medida de tiempo es tal que el dueño del reloj nunca necesita resetear el tiempo. Ajusta la hora y fecha mostrada según el usuario cruza zonas horarias y fronteras políticas. Por esta razón, *SatWatch* no tiene botones ni controles accesibles al usuario. Determina su locación usando satélites GPS y, como tal, sufre de las mismas limitaciones que cualquier otro dispositivo GPS (como la imposibilidad de determinar la locación en ciertos momentos del día en regiones montañosas). Durante períodos de suspensión, *SatWatch* asume que no se cruza una zona horaria o frontera política. Corrige su zona horaria en cuanto termine el periodo de suspensión.

SatWatch tiene una pantalla de dos líneas, en la de arriba, la hora (hora, minuto, segundo, zona horaria), y en la de abajo, la fecha (día, número, mes y año). La tecnología de pantalla utilizada permite que el usuario vea la hora fecha incluso bajo condiciones de poca luz.

Cuando cambia de frontera política, el dueño del reloj puede actualizar el software del reloj usando el dispositivo *WebifyWatch* (incluído con el reloj) y una computadora conectada a internet.

Los requisitos funcionales de arriba se enfocan solo en la posible interacción entre el *SatWatch* y su mundo exterior (su dueño, GPS, y *WebifyWatch*). La descripción de arriba no se concentra en ningún detalle de implementación (procesador, lenguaje, tecnología de pantalla).

Requisitos No Funcionales

Los **requisitos no funcionales** describen aspectos del sistema que no están relacionados directamente con el comportamiento funcional del sistema. Los requisitos no funcionales incluyen una amplia variedad de requisitos que se aplican a muchos aspectos diferentes del sistema, desde la usabilidad hasta el desempeño. El modelo *FRUPS+* usado por el "Proceso Unificado" [Jacobson, 1999] dicta las siguientes categorías de requisitos no funcionales:

- **Usabilidad:** es la facilidad con la que el usuario puede aprender a operar, introducir datos, e interpretar resultados de un sistema o componente. Los requisitos de usabilidad incluyen, por ejemplo, convenciones adoptadas por la interfaz de usuario, el alcance de ayuda en línea, y el nivel de documentación de usuario. Generalmente, los clientes abordan problemas de compatibilidad al necesitar que el desarrollador siga guías de interfaz de usuario sobre esquemas de color, logos y tipos de letra.
- **Confiabilidad:** es la habilidad de un sistema o componente para realizar la función requerida bajo condiciones establecidas por un cierto periodo de tiempo. Los requisitos de confiabilidad incluyen, por ejemplo, un tiempo promedio de falla aceptable y la habilidad de detectar fallas específicas o de soportar ciertos ataques a la seguridad. Más recientemente, esta categoría a menudo es reemplazada por la **confianza**, que es la propiedad de un sistema computacional que permite que la dependencia pueda ser colocada justificadamente en el servicio que ofrece. La confianza incluye confiabilidad, robustez (el grado en el que un sistema pude funcionar correctamente ante la presencia de entradas inválidas o condiciones de entorno estresantes), y seguridad (medida de la usencia de consecuencias catastróficas para el entorno).
- **Desempeño:** Los requisitos de desempeño se encargan de los atributos cuantificables del sistema, como el **tiempo de respuesta** (qué tan rápido reacciona

el sistema ante la entrada introducida por un usuario), **rendimiento** (cuánto trabajo puede realizar el sistema dentro de una cantidad específica de tiempo), **disponibilidad** (el grado en el que un sistema o componente es operacional y accesible al momento de necesitarlo) y, **certeza**.

- **Compatibilidad:** Los requisitos de compatibilidad se refieren a la facilidad de realización de cambios al sistema después de su implementación, incluyendo por ejemplo, **adaptabilidad** (la capacidad de cambiar el sistema para hacer frente a los conceptos adicionales de dominio de aplicación), **mantenibilidad** (la capacidad de cambiar el sistema para hacer frente a nuevas tecnologías o de reparar defectos), e **internacionalización** (la capacidad de cambiar el sistema para hacer frente a nuevas convenciones internar, como lenguajes, unidades, y número de formatos). El estándar ISO 9126 de calidad de software [ISO Std. 9126], parecido al modelo FRUPS+, reemplaza esta categoría con otras dos: **mantenibilidad** y **portabilidad** (la facilidad con la que el sistema o componente puede transferirse de un hardware o entorno de software a otro).

El modelo FRUPS+ brinda categorías adicionales de requisitos que típicamente se incluyen bajo la etiqueta general de requisitos no funcionales:

- Los **requisitos de implementación** son limitaciones en la implementación del sistema, incluyendo el uso de herramientas específicas, lenguajes de programación, o plataformas de hardware.
- Los **requisitos de interfaz** son limitaciones impuestas por sistemas externas, incluyendo sistemas antiguos/actuales e intercambio de formatos.
- Los **requisitos de operación** son limitaciones de administración y gestión del sistema en la configuración operativa.
- Los **requisitos de empaquetamiento** son limitaciones en la entrega del sistema (por ejemplo, limitaciones del medio de instalación para la inicialización del software).
- Los **requisitos legales** se encargan de los asuntos de licenciamiento, regulación, y certificación. Un ejemplo es que el software desarrollado para el gobierno federal de USA debe cumplir con la Sección 508 del Acta de Rehabilitación de 1972, requeriendo que sistemas informáticos del gobierno deben ser accesibles para las personas con discapacidades.

Los requisitos que caen en las categorías URPS son llamados requisitos de calidad del sistema. Los requisitos no funcionales que caen en las categorías de implementación, operaciones, interfaz, empaquetamiento, y legales. Los requisitos de calendario y presupuesto generalmente no se manejan como no funcionales, pues restringen atributos de los proyectos.

La figura 4-3 ilustra los requisitos no funcionales de *SatWatch*.

Requisitos de Calidad para SatWatch

- Cualquier usuario que sepa leer un reloj digital y entienda las abreviaciones de la zona horaria internacional debe ser capaz de usar el *SatWatch* sin un manual de usuario. [Requisito de Usabilidad]
- Como el *SatWatch* no tiene botones, no deberían haber fallas de software que necesiten del reseteo del reloj. [Requisito de Confiabilidad]
- *SatWatch* bebe mostrar la zona horaria correcta 5 minutos después de un periodo de suspensión de GPS. [Requisito de Desempeño]
- *SatWatch* debe medir el tiempo en 1/100 segundos por 5 años [Requisitos de Desempeño].
- *SatWatch* debe mostrar correctamente el tiempo en todas las 24 zonas horarias. [Requisito de Desempeño]
- *SatWatch* debe aceptar actualizaciones del software empotrado a través de la interfaz serial del *WebifyWatch* [Requisito de Compatibilidad]

Limitaciones de SatWatch

- Todo el software relacionado con *SatWatch*, incluyendo el software empotrado, será escrito en Java, para cumplir con las políticas actuales de la compañía. [Requisito de Implementación]
- *SatWatch* cumple con las interfaces física, eléctrica y de software definidas por *WebifyWatch* API 2.0. [Requisito de Interfaz].

Completitud, Coherencia, Claridad y Exactitud

Los requisitos se validan constantemente con el cliente y el usuario. La validación es un paso crítico en el proceso de desarrollo, dado que tanto el cliente como el desarrollador dependen de la especificación de requisitos. La validación de requisitos envuelve la revisión que la especificación sea completa, consistente, inequívoca y exacta. Es **completa**, si describe todos los posibles escenarios por los que puede pasar el sistema, incluyendo comportamiento excepcional (por ejemplo, todos los aspectos del sistema están representados en el modelo de requisitos). Es **consistente** si no se contradice a sí misma. La especificación de requisitos es **inequívoca** si se define exactamente un sistema (por ejemplo, no es posible interpretar la especificación de dos o más formas diferentes). Una especificación es **exacta** si representa acertadamente el sistema que el cliente necesita y que los desarrolladores intentan construir (por ejemplo, todo en el modelo de requisitos representa correctamente un aspecto del sistema que satisface al cliente y a los desarrolladores). Estas propiedades se ilustran en la tabla 4-1.

La **exactitud** y **completitud** de una especificación de requisitos son generalmente difíciles de establecer, especialmente antes de que exista el sistema. Dado que la especificación sirve como base contractual entre el cliente y los desarrolladores, debe ser cuidadosamente revisada por ambas

Tabla 4-1 Propiedades de la especificación revisadas durante la validación
Completa – Los requisitos describen todas las características de interés. *Ejemplo de falta de completitud*: La especificación del *SatWatch* no especifica el comportamiento fronterizo de cuando el usuario está dentro de los límites de precisión GPS de frontera de un Estado. *Solución:* Añadir un requisito funcional estableciendo que el tiempo mostrado por el *SatWatch* no debe cambiar más de una vez cada cinco minutos.
Consistente – Los requisitos de la especificación no se contradicen entre ellos. *Ejemplo de inconsistencia:* Un reloj sin falla alguna de software no tiene que proporcionar un mecanismo de actualización para descargar las nuevas versiones del software. *Solución:* Revisar uno de los requisitos conflictivos del modelo (por ejemplo, reformular el requisito sobre el reloj que no contiene ningún fallo, ya que no es verificable de todos modos).
Inequívoca – Un requisito no puede ser interpretado de dos maneras mutuamente excluyentes. *Ejemplo de ambigüedad:* La especificación del *SatWatch* se refiere a las zonas horarias y las fronteras políticas. ¿El *SatWatch* se encarga del horario de verano o no? *Solución:* Aclarar el concepto ambiguo para seleccionar uno de los fenómenos mutuamente excluyentes (por ejemplo, agregar un requisito para que *SatWatch* se encargue del horario de verano).
Correcta - Los requisitos describen las características del sistema y el entorno de interés para el cliente y el desarrollador, pero no describen otras características no deseadas. *Ejemplo de falta:* Hay más de 24 zonas horarias. Varios países y territorios (por ejemplo, la India) están adelantados media hora a la zona horaria de un país vecino.

partes. Adicionalmente, partes del sistema que representan un alto riesgo deben ser simuldas o construir un prototipo para demostrar su factibilidad u obtener retroalimentación del usuario. En el caso de *SatWatch*, descrito arriba, se creará una maqueta del reloj a partir de un reloj tradicional y los usuarios encuestados para recoger sus primeras impresiones. Un usuario puede hacer énfasis en que quiere que el reloj sea capaz de mostrar ambos formatos, el Americano y el Europeo.

Realismo, Verificabilidad y Trazabilidad

Tres propiedades deseables más de una especificación de requisitos son que debe ser realista, verificable, y trazable. La especificación de requisitos es **realista** si el sistema puede ser implementado dentro de las restricciones. Es **verificable** si, una vez que el sistema está construido, pueden diseñarse pruebas repetibles para demostrar que el sistema cumple con los requisitos de la especificación. Por ejemplo, un tiempo promedio de fallo de 100 años para el *SatWatch* sería difícil de verificar (suponiendo que sea realista en primer lugar). Los siguientes requisitos son ejemplos adicionales de requisitos no verificables:

- El producto deberá tener una buena interfaz de usuario – No se define "Buena".
- El producto deberá ser libre de errores – Requiere de un gran número de recursos para establecerlo.
- El producto deberá responder al usuario con un segundo la mayoría de las veces – No se define "la mayoría de las veces".

Una especificación de requisitos es **trazable** si cada requisito puede enlazarse a lo largo del desarrollo del software con sus funciones correspondientes en el sistema, y si cada función del sistema puede enlazarse de vuelta a su conjunto de requisitos correspondiente. La trazabilidad también incluye la habilidad de rastrear las dependencias entre requisitos, funciones del sistema, y los artefactos de diseño intermedios, incluyendo componentes del sistema, clases, métodos, y atributos objeto. La trazabilidad es muy importante para el desarrollo de pruebas y la evaluación de cambios. Al desarrollar pruebas, la trazabilidad le permite al encargado de la prueba evaluar la cobertura de un caso de prueba, es decir, identificar cuáles requisitos se prueban y cuáles no. Al evaluar cambios, la trazabilidad le permite al analista y a los desarrolladores identificar todos los componentes y funciones del sistema que se verán afectados por el cambio.

Ingeniería Greenfiled, Reingeniería, e Ingeniería de Interfaz.

Las actividades de la educción de requisitos pueden clasificarse en tres categorías, dependiendo del origen del requisito. En la **ingeniería greenfield**, el desarrollo empieza desde cero – no existen sistemas previos – así que los requisitos se obtienen de los usuarios y el cliente. Un proyecto greenfield nace por la necesidad de un usuario o la creación de un mercado.

SatWatch es un proyecto greenfield.

Un proyecto de **reingeniería** es el rediseño y re-implementación de un sistema existente, llevado a cabo por habilitadores de tecnología, o por los procesos de negocio [Hammer y Champy, 1993]. A veces se extiende la funcionalidad de los nuevos sistemas, pero el

propósito esencial del sistema sigue siendo el mismo. Los requisitos del nuevo sistema se extraen de un sistema existente.

Un proyecto de **ingeniería de interfaz** es el rediseño de la interfaz de usuario de un sistema existente. El sistema existente queda intacto, a excepción de su interfaz, que se ve rediseñada y re-implementada. Este tipo de proyecto es un proyecto de reingeniería en el que el sistema existente no puede descartarse sin traer con ello costos altos.

En ambas ingenierías, greenfield y reingeniería, los desarrolladores necesitan recolectar tanta información como sea posible del dominio de la aplicación. Esta información puede encontrarse en manuales procedimentales, documentación distribuida a nuevos empleados, el manual del sistema anterior, glosarios, notas y trucos desarrollados por los usuarios, y entrevistas con usuarios y clientes. Nótese que aunque las entrevistas con los usuarios son una herramienta invaluable, fallan en la recolección de información si no se hacen preguntas relevantes. Primero, los desarrolladores deben ganar conocimiento sólido del dominio de la aplicación antes de un acercamiento directo.

4.4 Actividades de la Educción de Requisitos

Las actividades de educción de requisitos asignan un planteamiento de problema a una especificación de requisitos que representamos como un conjunto de actores, escenarios y casos de uso. Discutiremos heurísticas y métodos para la obtención de los requisitos de los usuarios y el modelado del sistema en base a estos conceptos. Las actividades de educción de requisitos incluyen:

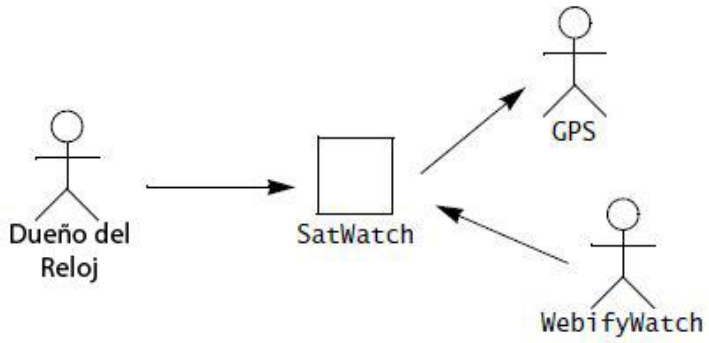
- Identificación de Actores (Sección 4.4.1)
- Identificación de Escenarios (Sección 4.4.2)
- Identificación de Casos de Uso (Sección 4.4.3)
- Refinamiento de Casos de Uso (Sección 4.4.4)
- Identificación de Relaciones entre Actores y Casos de Uso (Sección 4.4.5)
- Identificación de Objetos de Análisis Inicial (Sección 4.4.6)
- Identificación de Requisitos No Funcionales (Sección 4.4.7)

Los métodos descritos en esta sección se adaptaron de OOSE [Jacobson et al., 1992], el Proceso Unificado de Desarrollo de Software [Jacobson et al., 1999], y diseño basado en responsabilidad [Wirfs-Brock et al., 1990].

Identificación de Actores

Los actores representan entidades externas que interactúan con el sistema. Un actor pude ser humano o un sistema externo. En el ejemplo del *SatWatch*, el dueño del reloj, los satélites GPS, y el dispositivo serial *WebifyWatch* son actores (vea Figura 4-4). Todos ellos

intercambian información con el *SatWatch*. Sin embargo, nótese que todos ellos tienen interacciones específicas con *SatWatch*: el dueño del reloj usa y ve su reloj; el reloj monitorea la señal de satélites GPS; el *WbifyWatch* descarga nueva información al reloj. Los actores definen las clases de funcionalidad.



Considere un ejemplo más complejo, *FRIEND*, un sistema de información distribuida para el manejo de accidentes [Brugge et al., 1994]. Incluye muchos actores, como *FieldOfficer*, que representa a los oficiales de policía y de bomberos que acuden al incidente, y *Dispatcher*, el oficial de policía responsable de contestar llamadas al 911 y de asignar recursos al incidente.

FRIEND apoya a ambos actores, rastreando incidentes, recursos, y planes de trabajo. También tiene acceso a múltiples bases de datos, como una base de datos de materiales peligrosos y procedimientos de operaciones de emergencia. Los actores *FieldOfficer* y *Dispatcher* interactúan a través de diferentes interfaces: *FieldOfficer* accede a *FRIEND* a través de un asistente personal, *Dispatcher* a través de una estación de trabajo (vea Figura 4-5).

Los actores son abstracciones de rol y no necesariamente asignados directamente a personas. La misma persona puede ocupar el puesto de FieldOfficer o Dispatcher en momentos diferentes. Sin embargo, la funcionalidad a la que acceden es substancialmente diferente. Por esa razón, estos roles se modelan como dos actores diferentes.

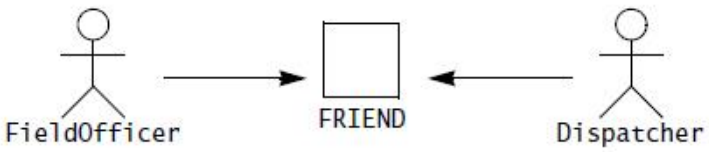


Figura 4-5 Los actores del sistema *FRIEND*. No solo *FieldOfficer* tiene acceso a diferente funcionalidad usan computadoras diferentes para acceder al sistema.

El primer paso de la educción de requisitos es la identificación de actores. Esto sirve tanto para definir los límites del sistema como para encontrar todas las perspectivas de las que los desarrolladores deben considerar el sistema.

Que considerar el sistema. Cuando el sistema se implementa en una organización existente (por ejemplo una empresa) por lo general la mayoría de los actores existen antes de que el sistema se haya desarrollado: ellos corresponden a los roles de la organización.

Durante las etapas iniciales de identificación del actor, es difícil distinguir los actores de los objetos.

Por ejemplo, un subsistema de base de datos puede ser un tiempo un actor, mientras que en otros casos puede ser parte del sistema. Tenga en cuenta que una vez que se define la frontera del sistema, no hay problemas para distinguir entre los actores y los componentes del sistema tales como objetos o subsistemas. Los actores se encuentran fuera de los límites del sistema, ellos son externos. Los subsistemas y los objetos están dentro de la frontera del sistema, ellos son internos, por lo tanto, cualquier sistema de software externo utilizando el sistema a desarrollar es un actor. En la identificación de los actores, los desarrolladores pueden hacer las siguientes preguntas:

Preguntas para la identificación de los actores:

- ¿Qué grupos de usuarios son compatibles con el sistema para realizar su trabajo?
- ¿Qué grupos de usuarios ejecutan las principales funciones del sistema?
- ¿Qué grupos de usuarios realizan funciones secundarias, como el mantenimiento y la administración?
- ¿Con que sistema de hardware o software externo interactuará el sistema?

En el ejemplo AMIGO, estas preguntas conducen a una larga lista de actores potenciales: bombero, oficial de policía, despachador, investigador, alcalde, gobernador, base de datos de materiales peligrosos de la EPA, administrador del sistema, y así sucesivamente. Entonces necesitamos para consolidar esta lista en un pequeño número de actores, que son diferentes desde el punto de vista de la utilización del sistema. Por ejemplo, un bombero y un oficial de policía puedo compartir la misma interfaz para el sistema, ya que están involucrados con un solo incidente en el campo. Un despachador, por otra parte, gestiona múltiples incidentes concurrentes y requiere el acceso a una mayor cantidad de información. El alcalde y el gobernador no es probable que interactúen directamente con el sistema, si no que utilicen los servicios de un operador entrenado en su lugar. Una vez

que se identifican los actores, el siguiente paso en la actividad de la obtención de requisitos es determinar la funcionalidad que será accesible a cada actor. Esta información puede ser extraída por medio de escenarios y formalizada mediante casos de uso.

Escenarios de Identificación

Un escenario es "una descripción narrativa de lo que se hace y la experiencia en su intento de hacer uso de los sistemas informáticos y aplicaciones" [Carroll, 1995]. Un escenario es un concreto, centrado, descripción informal de una sola característica del sistema desde el punto de vista de un solo actor.

Los escenarios pueden no (y no tienen por objeto) reemplazar los casos de uso, ya que se centran en casos específicos y eventos concretos (en lugar de completar y descripciones generales). Sin embargo, los escenarios de mejorar la obtención de requisitos, proporcionando una herramienta que sea comprensible para los usuarios y clientes.

Figura 4-6 es un ejemplo de escenario para el sistema AMIGO, un sistema de información para respuesta a incidentes. En este escenario, un oficial de policía informa de un incendio y una Dispatcher inicia la respuesta a incidentes. Tenga en cuenta que este escenario es de hormigón, en el sentido de que describe una sola

---

Nombre del escenario	Almacén en llamas.
Actor participante	Bob: Alice: oficial de campo
Instancias	John: Despachador

Flujo de eventos

1. Bob, conduciendo por la calle principal en su patrulla, avisos de humo que sale de un almacén. Su compañera, Alice, activa la función de "Informe de emergencia" desde su portátil AMIGO.

2. Alice entra en la dirección del edificio, una breve descripción de su ubicación (es decir, en la esquina noroeste), y un nivel de emergencia. Además de una unidad de fuego, que solicita varias

unidades de paramédicos en la escena, dado que el área de parecer ser relativamente ocupada. Ella confirma su entrada y espera un acuse de recibo.

3. John, el Dispatcher, se alerta a la emergencia de un pitido de su estación de trabajo. Pasa revisa la información presentada por Alice y reconoce el informe. El asigna una unidad de fuego y dos unidades de paramédicos al lugar del incidente y envía su hora estimada de llegada (ETA) a Alice.

4. Alice recibe el reconocimiento y la ETA.

Figura 4-6

Instancia. No intenta describir todas las posibles situaciones en las que un incidente de fuego está reportado. En particular, los escenarios no pueden contener descripciones de decisiones. Para describir la resultado de una decisión, se necesitarían dos escenarios, uno para el camino "verdadero", y otro para la ruta de "falso".

Los escenarios pueden tener muchos usos diferentes durante la obtención de requisitos y durante otras actividades del ciclo de vida. A continuación se muestra un número seleccionado de tipos de escenarios tomados de [Carroll, 1995]:

• Tal Cual escenarios describen una situación actual. Durante la reingeniería, por ejemplo, el sistema actual se entiende al observar a los usuarios y la descripción de sus acciones como escenarios. Estos escenarios pueden ser validados por la exactitud y la precisión con la los usuarios.

• Escenarios Visionar describen un sistema futuro. Escenarios visionarios se utilizan tanto como punto en el espacio de modelado por los desarrolladores, ya que refinan sus ideas del futuro sistema y como un medio de comunicación para obtener los requisitos de los usuarios. Escenarios visionarios puede ser visto como un prototipo de bajo costo.

• Escenarios de evaluación describen las tareas del usuario contra el que el sistema se va a evaluar.

El desarrollo colaborativo de escenarios de evaluación por los usuarios y los desarrolladores también mejora la definición de la funcionalidad probada por estos escenarios.

• Escenarios de entrenamiento son tutoriales utilizados para la introducción de los nuevos usuarios en el sistema. Estos se pasó a paso las instrucciones destinadas a mano sostenga el usuario a través de las tareas comunes.

En la obtención de requisitos, los desarrolladores y los usuarios escriben y perfeccionan una serie de escenarios con el fin de obtener una comprensión compartida de lo que debe ser el sistema. Inicialmente, cada escenario puede ser de alto nivel e incompleta, ya que el escenario es warehouseOnFire. Las siguientes preguntas se pueden usar para identificar escenarios.

Preguntas para la identificación de escenarios
• ¿Cuáles son las tareas que el actor quiere que el sistema realice?
• ¿Qué información de acceso actor?
¿Quien crea que los datos? ¿Puede ser modificado o eliminado? ¿Por quién?
• ¿Qué cambios externos necesita el actor para informar al sistema acerca de? ¿Con qué frecuencia? ¿Cuándo?
• ¿Qué eventos necesita el sistema para informar sobre el actor? ¿Con lo que la latencia?

Los desarrolladores utilizan los documentos existentes sobre el dominio de aplicación para responder a estas preguntas. Estos documentos incluyen los manuales de usuario de los sistemas anteriores, manuales de procedimientos, estándares de la compañía, las notas de usuario y hojas de trucos, de usuario y de entrevistas con clientes. Desarrolladores siempre deben escribir escenarios utilizando términos del dominio de aplicación, en lugar de sus propios términos. Como desarrolladores de obtener una mayor comprensión del dominio de aplicación y las posibilidades de la tecnología disponible, que de forma iterativa e incremental refinar los escenarios para incluir cantidades crecientes de detalle. Dibujo de usuario de interfaz de maquetas a menudo ayuda a encontrar omisiones en la descripción y la construcción de una imagen más concreta del sistema. En el ejemplo AMIGO, identificamos cuatro escenarios que abarcan el tipo de tareas que se espera que el sistema de apoyo:

WarehouseOnFire (Figura 4-6): Un incendio es detectado en una bodega, dos oficiales de campo llegan a la escena y solicitar recursos.

• Fenderbender: Un accidente de coche sin bajas se produce en la carretera. Los agentes de policía documentar el incidente y gestionar el tráfico, mientras que los vehículos dañados son retirados por la grúa.

• catInATree : Un gato se ha quedado atascado en un árbol. Un camión de bomberos se llama para recuperar el gato. Debido a que el incidente es de baja prioridad, el camión de bomberos necesita tiempo para llegar a la escena. Mientras tanto, el dueño del gato impaciente sube al árbol, cae, y se rompe una pierna, lo que requiere una ambulancia para ser enviados.

• Terremoto: Un terremoto sin precedentes perjudica gravemente los edificios y carreteras, que abarcan múltiples incidentes y que desencadenan la activación de un plan de operaciones de emergencia en todo el estado. El gobernador se notifica. Daño de camino dificulta la respuesta a incidentes.

El énfasis para los desarrolladores durante la identificación de la identificación y el escenario el actor es entender el dominio de aplicación. Esto da lugar a una comprensión compartida del alcance del sistema y de los procesos de trabajo de los usuarios que deben apoyarse. Una vez que los desarrolladores se han identificado y descrito los actores y escenarios, formalizan escenarios en los casos de uso.

Identificando caso de uso

Un escenario es una instancia de un caso de uso, es decir, un caso de uso especifica todos los escenarios posibles para una determinada funcionalidad. Un caso de uso es iniciado por un actor. Después de su iniciación, un caso de uso puede interactuar con otros actores, también. Un caso de uso representa un flujo completo de eventos a través del sistema en el sentido de que describe una serie de interacciones relacionadas que resultan de su iniciación.

Figura 4-7 muestra el caso de uso ReportEmergency de que el escenario warehouseOnFire (ver Figura 4-6) es un ejemplo. El actor FieldOfficer inicia este caso de uso mediante la activación de la función "Informe de emergencia" de AMIGO. El caso de uso termina cuando el actor FieldOfficer recibe un reconocimiento de que un incidente ha sido creado. Los pasos en el flujo de los acontecimientos se sangran para denotar que inicia la etapa. Pasos 1 y 3 son iniciados por el actor, mientras que los pasos

Nombre del caso de uso ReportEmergency

Actores participantes Iniciado por FieldOfficer

Se comunica con Dispatcher

Flujo de eventos

1. El FieldOfficer activa la función "Informe de emergencia" de su terminal.
2. AMIGO responde mediante la presentación de un formulario a la FieldOfficer.
3. El FieldOfficer completa el formulario seleccionando el nivel de emergencia, tipo,
Ubicación, y breve descripción de la situación. El FieldOfficer también describe las posibles respuestas a la situación de emergencia. Una vez que el formulario es completado, el FieldOfficer envía el formulario.
4. AMIGO recibe el formulario y notifica al despachador.
5. El Dispatcher revisa la información presentada y crea un incidente en la base de datos mediante la invocación del caso de uso OpenIncident. El despachador selecciona una respuesta y reconoce el informe.
6. AMIGO muestra el reconocimiento y la respuesta seleccionada a la FieldOfficer.
Condiciones de entrada • El FieldOfficer se registra en AMIGO.

Condiciones de entrada • El FieldOfficer se registra en AMIGO.
Condiciones de salida • El FieldOfficer ha recibido un acuse de recibo y la respuesta seleccionada del Dispatcher, o
• El FieldOfficer ha recibido una explicación que indique por qué la transacción no se pudo procesar.
Calidad requisitos
• El informe del FieldOfficer se reconoce dentro de los 30 segundos.
• La respuesta seleccionada llega a más de 30 segundos después de que se envíe por el Dispatcher.

Figura 4-7 Un ejemplo de un caso de uso, ReportEmergency. Bajo ReportEmergency, la columna de la izquierda denota las acciones de los actores y la columna de la derecha denota las responsabilidades del sistema.

2 y 4 son iniciadas por el sistema. Este caso de uso es general y abarca una amplia gama de escenarios. Por ejemplo, el caso de uso ReportEmergency también podría aplicarse a la situación Fenderbender. Los casos de uso pueden ser escritos en diferentes niveles de detalle como en el caso de los escenarios. Generalizando escenarios e identificar los casos de uso de alto nivel que el sistema debe admitir permite a los desarrolladores para definir el alcance del sistema. En un principio, los casos de uso Nombre desarrolladores, la fijación a los actores que inician, y proporcionan una descripción de alto nivel del caso de uso como en la Figura 4-7. El nombre de un caso de uso debe ser una frase verbal que denota lo que el actor está tratando de lograr. La frase verbal "Informe de emergencia " indica que un actor está tratando de informar sobre una emergencia en el sistema (y por lo tanto, al actor Dispatcher). Este caso de uso no se llama " Emergencia Record" porque el nombre debe reflejar el punto de vista del actor, no el sistema. Tampoco se le llama "El intento de reportar una emergencia " porque el nombre debe reflejar el objetivo del caso de uso, y no la actividad real. Colocación de casos de uso para iniciar actores permite a los desarrolladores para aclarar las funciones de los diferentes usuarios. A menudo, al centrarse en que inicia cada caso de uso, los desarrolladores a identificar nuevos actores que se han pasado por alto previamente.

Al describir un caso de uso implica la especificación de cuatro campos. Al describir las condiciones de entrada y de salida de un caso de uso permite a los desarrolladores a entender las condiciones en las que se invoque un caso de uso, y el impacto del caso de uso en el estado del medio ambiente y del sistema. Mediante el examen de las condiciones de entrada y de salida de los casos de uso, los desarrolladores pueden determinar si es posible que falten los casos de uso. Por ejemplo, si un caso de uso requiere que el plan de operaciones de emergencia frente a los terremotos se debe activar, la especificación de requisitos debe proporcionar también un caso de uso para la activación de este plan. Describir el flujo de eventos de un caso de uso permite a los desarrolladores y clientes para discutir la interacción entre los actores y el sistema. Esto da lugar a muchas decisiones sobre el límite del sistema, es decir, sobre decidir qué acciones se llevan a cabo por las acciones de actores y que se llevan a cabo por el sistema. Por último, la descripción de la calidad requisitos relacionados con un caso de uso permite a los desarrolladores para obtener los requisitos no funcionales en el contexto de una funcionalidad específica. En este libro, nos centramos en estos cuatro campos para describir casos de uso, ya que describen los aspectos más esenciales de un caso de uso. En la práctica, muchos adicional campos se pueden agregar para describir un flujo excepcional de eventos, reglas e invariantes que el uso caso deberá respetar durante el curso de los acontecimientos.

Escribir casos de uso es un arte. Un analista aprende a escribir mejores casos de uso con experiencia. En consecuencia, diferentes analistas tienden a desarrollar diferentes estilos, que puede hacer que sea difícil producir una especificación de requisitos consistentes. Para abordar el tema de aprender a escribir casos de uso y la forma de garantizar la coherencia entre los casos de uso de una especificación de requisitos, analistas adoptan una guía de escritura de casos de uso. Figura 4-8 es una guía de escritura sencilla adaptada de [Cockburn, 2001] que se puede utilizar para los escritores noveles de casos de uso. Figura 4-9 proporciona un ejemplo de un caso de mal uso que viole las directrices por escrito de varias maneras.

El caso de uso ReportEmergency en la Figura 4-7 puede ser lo suficientemente ilustrativa para describir cómo AMIGO apoya emergencias de información y para obtener comentarios generales por parte del usuario, pero sí no proporcionar suficientes detalles para una especificación de requisitos. A continuación, se discute cómo los casos de uso son refinado y detallado.

Guía de la escritura de casos de uso simple

• Los casos de uso deben ser nombradas con frases verbales. El nombre del caso de uso se debe indicar lo que el usuario está tratando de lograr (por ejemplo, ReportEmergency, OpenIncident).

• Los actores deben ser nombradas con sintagmas nominales (por ejemplo, FieldOfficer, Dispatcher, víctima).

• El límite del sistema debe ser clara. Pasos realizados por el actor y pasos logrado por el sistema debe distinguirse (por ejemplo, en la Figura 4-7, las acciones del sistema se aplica sangría a la derecha).

• Pasos de casos de uso en el curso de los acontecimientos deben formularse en la voz activa. Esto hace que explícita logrado el paso.

• La relación causal entre los pasos sucesivos debe ser clara.

• Un caso de uso debe describir una transacción de usuario completa (por ejemplo, el caso de uso ReportEmergency describe todos los pasos entre la iniciación de la notificación de emergencia y la recepción de un acuse de recibo).

• Las excepciones deben describirse por separado.

• Un caso de uso no se debe describir la interfaz de usuario del sistema. Esto quita el foco de la medidas reales realizadas por el usuario y puede abordarse mejor con maquetas visuales (por ejemplo, la ReportEmergency sólo se refiere a la función de "Informe de emergencia", no en el menú, el botón, ni el comando real que corresponde a esta función).

• Un caso de uso no debe superar las dos o tres páginas de extensión. De lo contrario, el uso de incluir y extender relaciones que se descomponen en los casos de uso más pequeños, como se explica en la Sección 4.4.5.

Nombre del caso de uso	Accidente
Actor iniciante	Iniciado por FieldOficer
Flujo de eventos	1.- El FieldOfficer reporta el accidente 2.-Una ambulancia es asignada. 3.- El Dispatcher es notificado cuando la ambulancia llega al sitio.

Refinación Casos de Uso

Figura 4-10 es una versión refinada del caso de uso ReportEmergency. Se ha ampliado para incluir información sobre el tipo de incidentes conocidos AMIGO e interacciones detalladas que indican cómo el Dispatcher reconoce la FieldOfficer.

Nombre del caso de uso	Reporte de emergencia
Actor iniciante	Iniciado por FieldOficer
Flujo de eventos	1. El FieldOfficer activa la función "Informe de emergencia" de su terminal. 2. AMIGO responde mediante la presentación de un formulario para que el oficial. *El formulario incluye un menú de tipo de emergencia (emergencia general, el fuego, el transporte) y la ubicación, descripción del incidente, solicitud de recursos, y los campos de materiales peligrosos*.

3. El FieldOfficer completa el formulario especificando mínimamente la situación de emergencia campos de tipo y descripción. El FieldOfficer también puede describir las posibles respuestas a la situación de emergencia y solicitar recursos específicos. Una vez completado el formulario, el FieldOfficer envía el formulario.

4. AMIGO recibe el formulario y notifica al despachador por un cuadro de diálogo emergente.

5. El Dispatcher revisa la información presentada y crea un incidente en la base de datos mediante la invocación del caso de uso OpenIncident. Toda la información contenida en el formulario de la FieldOfficer se incluye automáticamente en el incidente. El Dispatcher selecciona una respuesta mediante la asignación de recursos para el Incidente (con los AllocateResources utilizan caso) y reconoce el informe de emergencia mediante el envío de un mensaje corto al FieldOfficer.

6. AMIGO muestra el reconocimiento y la respuesta seleccionada a la FieldOfficer.

Condición de entrada

- …

Figura 4-10 Descripción refinado para el caso de uso ReportEmergency. Adiciones hicieron hincapié en cursiva.

El uso de escenarios y casos de uso para definir la funcionalidad del sistema tiene como objetivo la creación de requisitos que son validados por el usuario al principio del desarrollo. Como se inicia el diseño y la implementación del sistema, el costo de cambiar la especificación de requisitos y la adición de nuevas funcionalidades aumentos imprevistos. Aunque los requisitos de cambio hasta tarde en el desarrollo, los desarrolladores y los usuarios deben esforzarse para hacer frente a la mayoría de los problemas de los requisitos antes de tiempo. Esto implica muchos cambios y mucha validación durante la obtención de requisitos. Tenga en cuenta que los casos de uso de muchos se vuelven a escribir varias veces, otros se perfeccionaron sustancialmente, y sin embargo otros

completamente caídos. Para ahorrar tiempo, gran parte de los trabajos de exploración se puede hacer uso de los escenarios y la interfaz de usuario de maquetas.

Las siguientes heurísticas pueden ser utilizadas para los escenarios de escritura y casos de uso:

Heurística para el desarrollo de escenarios y casos de uso

• Usar escenarios para comunicarse con los usuarios y para validar la funcionalidad.

• En primer lugar, precisar un solo escenario para entender los supuestos de los usuarios sobre el sistema. El usuario puede estar familiarizado con sistemas similares, en cuyo caso, la adopción de los convenios específicos para la interfaz de usuario podría hacer que el sistema sea más utilizable.

• A continuación, definir muchos escenarios no muy detallados para definir el alcance del sistema. Validar con el usuario.

• Utilice maquetas tan sólo un apoyo visual, el diseño de la interfaz de usuario debe ocurrir como una tarea independiente después de la funcionalidad es suficientemente estable.

• presentar al usuario con múltiples y muy diferentes alternativas (en oposición a la extracción de una sola alternativa por parte del usuario). La evaluación de las diferentes alternativas se amplía el horizonte del usuario. La generación de diferentes alternativas obliga a los desarrolladores a " pensar fuera de la caja. "

• Detalle de un amplio corte vertical cuando el alcance del sistema y las preferencias de los usuarios se conocen bien. Validar con el usuario.

El objetivo de esta actividad es el completo y correcto. Desarrolladores identifican funcionalidad no cubierta por los escenarios, y documentarla por los casos de uso de refinación o escribir otros nuevos. Desarrolladores describen casos rara vez ocurren y el manejo de excepciones como se ve por los actores. Considerando que la identificación inicial de casos de uso y actores centró en el establecimiento de los límites del sistema, el refinamiento de los casos de uso se obtiene cada vez más detalles sobre las características proporcionadas por el sistema y las restricciones asociadas con ellos. En particular, los siguientes aspectos de los casos de uso, ignorado inicialmente, se detallan en el refinamiento:

• Los elementos que son manipuladas por el sistema son detallada. En la Figura 4-10, hemos añadido detalles acerca de los atributos de la forma de notificación de emergencia y los tipos de incidentes.

• Se especifican la secuencia de bajo nivel de las interacciones entre el actor y el sistema.
En la Figura 4-10, hemos añadido información sobre cómo el Dispatcher genera un acuse de recibo mediante la selección de recursos.
• Se especifican los derechos de acceso (que los actores pueden invocar qué casos de uso).
• Excepciones que faltan son identificados y su manejo especificadas.
• La funcionalidad común entre los casos de uso se incluyen como factores fuera.
En la siguiente sección, se describe cómo reorganizar los actores y casos de uso con las relaciones, que aborda los últimos tres puntos por encima de las balas.

Identificación de relaciones entre los actores y casos de uso

Incluso los sistemas de tamaño medio tienen muchos casos de uso. Las relaciones entre actores y casos de uso permitir a los desarrolladores y usuarios para reducir la complejidad del modelo y aumentar su comprensibilidad. Utilizamos las relaciones de comunicación entre actores y casos de uso para describir el sistema en capas de funcionalidad. Utilizamos extender relaciones para separar excepcional y flujos comunes de eventos. Utilizamos incluir relaciones para reducir la redundancia entre los casos de uso.
Relaciones de comunicación entre los actores y casos de uso relaciones de comunicación entre actores y casos de uso representan el flujo de información durante el caso de uso. El actor que inicia el caso de uso debe ser distinguido de los otros actores con los que se comunica el caso de uso. Al especificar que el actor puede invocar un caso de uso específico, también especifica implícitamente el que los actores no pueden invocar el caso de uso.
Del mismo modo, mediante la especificación de los cuales los actores se comunican con un caso de uso específico, se especifica que actores pueden acceder a informaciones específicas y cuáles no. Por lo tanto, mediante la documentación de iniciación y relaciones de comunicación entre los actores y casos de uso, que especificar el control de acceso para el sistema en un nivel burdo .
Las relaciones entre actores y casos de uso se identifican cuando los casos de uso son identificados. La figura 4-11 ilustra un ejemplo de las relaciones de comunicación en el caso de la sistema AMIGO. Los « iniciar » denota el inicio del caso de uso por un actor, y la « participación » denota que un actor (que no se inicie el caso de uso) se comunica con el caso de uso .

Extender las relaciones entre casos de uso
Un caso de uso se extiende a otro caso de uso si el caso de uso extendido puede incluir el comportamiento de la extensión bajo ciertas condiciones. En el ejemplo AMIGO, asumir que la conexión entre la estación y la estación FieldOfficer Dispatcher se rompe mientras que el

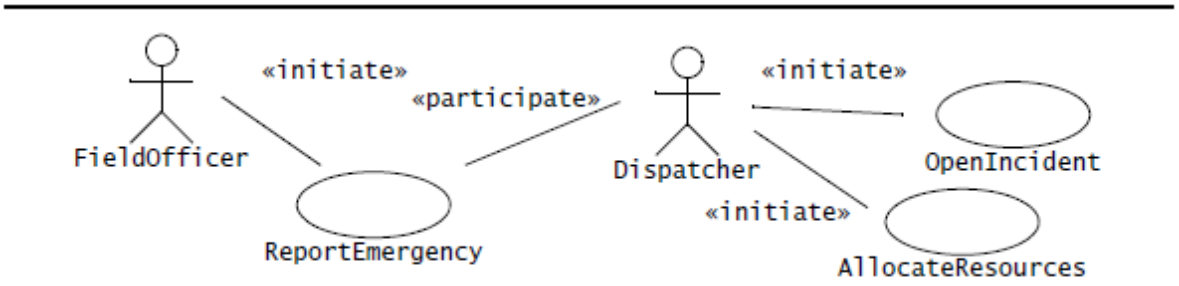


Figura 4-11 Ejemplo de relaciones de comunicación entre los actores y casos de uso en FRIEND (uso UML diagrama de casos). El FieldOfficer inicia el caso de uso ReportEmergency y el Dispatcher inicia la OpenIncident y AllocateResources casos de uso. FieldOfficers no pueden abrir directamente un incidente o asignar los recursos por su cuenta.

FieldOfficer está llenando la forma (por ejemplo, el coche de la FieldOfficer entra en un túnel). La estación FieldOfficer tiene que notificar a la FieldOfficer que su forma no se entregó y qué medidas debe tomar. El caso de uso ConnectionDown se modela como una extensión de ReportEmergency (véase la figura 4-12). Las condiciones en que se inicia el caso de uso ConnectionDown se describen en ConnectionDown en oposición a ReportEmergency. La separación de los flujos excepcionales y opcionales de los acontecimientos del caso de uso base tiene dos ventajas. En primer lugar, el caso de uso base se hace más corta y más fácil de entender. En segundo lugar, el caso común se distingue del caso excepcional, que permite a los desarrolladores para tratar cada tipo de funcionalidad diferente (por ejemplo, optimizar el caso común de tiempo de respuesta, optimizar el caso excepcional de solidez). Tanto el caso de uso extendido y las extensiones son casos de uso completos propios. Cada uno de ellos debe tener las condiciones de entrada y de fin y ser comprensible por el usuario como un todo independiente.

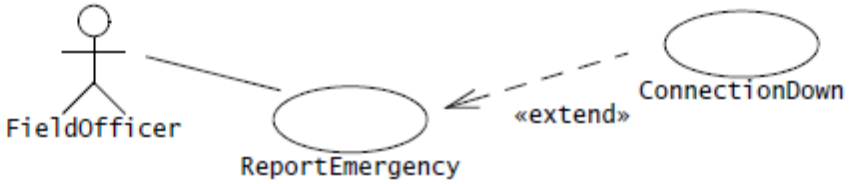


Figura 4-12 Ejemplo de uso de extender la relación (diagrama de uso UML caso). ConnectionDown extiende el caso de uso ReportEmergency. El caso de uso ReportEmergency hace más corto y exclusivamente centrado en la información de emergencia.

Incluya las relaciones entre casos de uso

Los despidos entre los casos de uso pueden tener en cuenta utilizando incluir relaciones. Supongamos, por ejemplo, que un despachador necesita consultar el mapa de la ciudad al abrir un incidente (por ejemplo, para evaluar qué áreas están en riesgo en caso de incendio) y en la asignación de recursos (por ejemplo, para encontrar los recursos que están más cerca del incidente). En este caso, el caso de uso ViewMap describe el flujo de eventos necesaria cuando se observa el mapa de la ciudad y es utilizado tanto por el OpenIncident y los AllocateResources casos de uso (Figura 4-13).

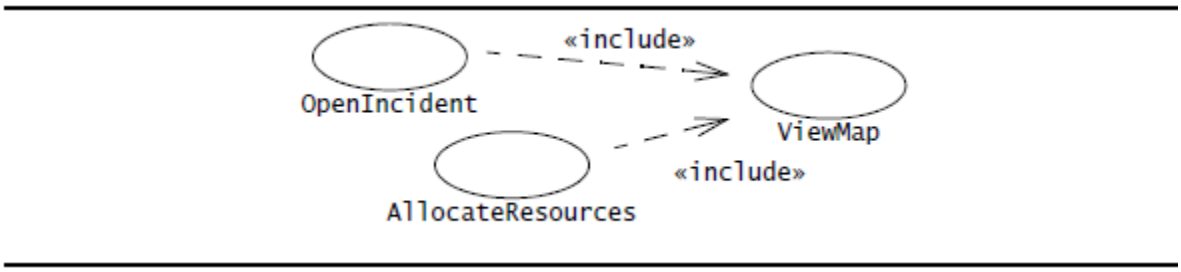


Figura 4-13 Ejemplo de incluir las relaciones entre los casos de uso. ViewMap describe el flujo de los acontecimientos para ver un mapa de la ciudad (por ejemplo, desplazamiento, zoom, búsqueda por nombre de la calle) y se utiliza tanto por OpenIncident y AllocateResources casos de uso.

Factorizar el comportamiento compartido de casos de uso tiene muchos beneficios, incluyendo descripciones más cortas y un menor número de redundancias. El Comportamiento sólo debe tenerse en cuenta en un caso de uso separado, si es compartida entre dos o más casos de uso. La fragmentación excesiva de la especificación de requisitos a través de un gran número de casos de uso hace que la especificación confunda a los usuarios y clientes.

Extender, incluir frente relaciones

Incluir y extender son construcciones similares, e inicialmente puede no estar claro para el desarrollador cuándo utilizar cada uno [Jacobson et al., 1992]. La distinción principal entre estas construcciones es la dirección de la relación. Para “incluir” las relaciones, el evento desencadenante del objetivo (es decir, “incluido”) caso de uso se describe en el flujo del evento del caso de uso de fuente. Para extender las relaciones, el hecho desencadenante de la fuente (es decir, que se “extiende”) casos de uso se describe en el caso de uso de origen como condición previa. En otras palabras, para “incluir” las relaciones, cada caso de

uso incluyendo deberá especificar donde se debe invocar el caso de uso “incluido”. Para extender las relaciones, sólo el caso de uso que “extiende” especifica qué casos de uso están extendidos. Por lo tanto, un comportamiento que está fuertemente ligado a un evento y que sólo se da en un número relativamente pequeño de casos de uso debe estar representados con una relación incluida. Este tipo de comportamiento por lo general incluyen las funciones normales del sistema que se pueden utilizar en varios lugares (por ejemplo, la visualización cartográfica, especificando un nombre de archivo, la selección de un elemento).
Por el contrario, un comportamiento que puede ocurrir en cualquier momento o cuya ocurrencia se puede especificar más fácilmente como una condición de entrada debe estar representada con una relación de extensión. Este tipo de comportamiento incluye situaciones excepcionales (por ejemplo, invocando la ayuda en línea, la cancelación de una transacción, que se ocupan de un fallo en la red).

Figura 4-14 muestra el ejemplo ConnectionDown descrito con una relación incluir (columna de la izquierda) y con una relación de extensión (columna derecha). En la columna de la izquierda, tenemos que insertar el texto en dos lugares en el flujo del evento donde se puede invocar el caso de uso ConnectionDown.

Además, si situaciones excepcionales adicionales se describen (por ejemplo, una función de ayuda en la Estación FieldOfficer), el caso de uso ReportEmergency tendrá que ser modificada y se convertirá llena de condiciones. En la columna de la derecha, tenemos que describir sólo las condiciones en las que se invoca el caso de uso excepcional, que pueden incluir un gran número de casos de uso (por ejemplo, "cualquier caso de uso en las que se pierde la conexión entre el FieldOfficer y el Dispatcher").

Por otra parte, las situaciones excepcionales que se pueden añadir sin modificar el caso de uso base (por ejemplo, ReportEmergency). La posibilidad de ampliar el sistema sin modificar las partes existentes es fundamental, ya que nos permite asegurar que el comportamiento original se conserva intacto. La distinción entre “incluir” y “extender” es un problema de documentación: utilizando el tipo correcto de la relación se reducen las dependencias entre los casos de uso, reduce la redundancia, y disminuye la probabilidad de introducir errores cuando los requisitos cambian. Sin embargo, el impacto sobre otras actividades de desarrollo es mínima.

Heurística para ampliar e incluyen las relaciones

• Uso extender en las relaciones de comportamiento excepcional, opcional, o casi nunca-ocurrentes. Un ejemplo de comportamiento que ocurre rara vez es la ruptura de un recurso (por ejemplo, un camión de bomberos). Un ejemplo de comportamiento opcional es la notificación de los recursos cercanos que responden a un incidente no relacionado.
• Use incluyen relaciones de comportamiento que se comparte entre dos o más casos de uso.
• Sin embargo, usar la discreción en la aplicación de las dos heurísticas anteriores y no overstructure el modelo de casos de uso. Pocos casos de uso prolongado (por ejemplo, dos páginas de largo) son más fáciles de entender y de revisión de muchos cortos (por ejemplo, diez líneas de largo).

En resumen, las siguientes heurísticas pueden ser utilizadas para la selección de una relación de extensión o un incluir.

Obtención de requerimientos actividades

En todos los casos, el propósito de la adición de incluir y extender las relaciones es para reducir o eliminar redundancias con el modelo de casos de uso, eliminando así posibles inconsistencias.

Identificación de Objetos análisis inicial

Uno de los primeros obstáculos de desarrolladores y usuarios se encuentran cuando empiezan a colaborar con cada uno otro se difiere la terminología. Aunque los desarrolladores finalmente aprenden la terminología de los usuarios, este problema es probable que se encuentren de nuevo cuando se añaden nuevos desarrolladores para el proyecto.
Los malentendidos son el resultado de los mismos términos que se utilizan en diferentes contextos y con diferentes significados.

Para establecer una terminología clara, los desarrolladores identifican los objetos que participan en cada caso de uso. Los desarrolladores deben identificar, nombrar y describir sin ambigüedad y cotejarlos en un glosario.

3. este glosario constituye el primer paso para el análisis, lo que nos discutir en el próximo capítulo.

El glosario se incluye en la especificación de requisitos y, más tarde, en los manuales de usuario.
Los desarrolladores mantienen el glosario hasta la fecha como la especificación de requisitos evoluciona. Los beneficios del glosario son muchos: nuevos desarrolladores están expuestos a un conjunto coherente de definiciones, un solo término se utiliza para cada concepto (en lugar de un término desarrollador y un término de usuario), y cada término tiene un significado oficial preciso y claro. La identificación de objetos que participan resulta en el modelo inicial objeto de análisis. La identificación de los objetos que participan en la obtención de requisitos sólo constituye un primer paso hacia el modelo completo objeto de análisis. El modelo de análisis completo por lo general no se utiliza como en los medios de comunicación entre los usuarios y los desarrolladores, ya que los usuarios están familiarizados con conceptos de orientación a objetos. Sin embargo, la descripción de los objetos (es decir, las definiciones de la términos en el glosario) y sus atributos son visibles para los usuarios y revisado. Se describe en detalle el perfeccionamiento del modelo de análisis en el capítulo 5, Análisis. Heurística para ampliar e incluyen las relaciones:

• Usar extender las relaciones de comportamiento excepcional, opcional, o casi nunca - ocurrentes. Un ejemplo de comportamiento raramente ocurre es la ruptura de un recurso (por ejemplo, un camión de bomberos). Un ejemplo de comportamiento opcional es la notificación de los recursos cercanos que responden a un incidente no relacionado.
• Usar incluyen relaciones de comportamiento que se comparte entre dos o más casos de uso.
• Sin embargo, usar la discreción en la aplicación de las dos heurísticas anteriores y no overstructure el uso modelo de caso. Pocos casos de uso más largos (por ejemplo, dos páginas de largo) son más fáciles de entender y revisión de muchos cortos (por ejemplo, diez líneas largas).

ReportEmergency (incluir la relación)

El FieldOfficer completa el formulario seleccionando el nivel de emergencia, el tipo, la ubicación y la descripción breve de la situación. El FieldOfficer también describe las posibles respuestas a la situación de emergencia. Una vez completado el formulario, el FieldOfficer envía el formulario, y en ese momento, el despachador se notifica.
Si la conexión con el despachador se rompe, se utiliza el caso de uso ConnectionDown.
4. Si la conexión está todavía viva, el distribuidor revisa la información presentada y crea un incidente en la base de datos mediante la invocación del caso de uso OpenIncident. El Dispatcher selecciona una respuesta y reconoce el informe de emergencia.

Si se interrumpe la conexión, se utiliza el caso de uso ConnectionDown. ReportEmergency (extender relación)
. El FieldOfficer completa el formulario seleccionando el nivel de emergencia , el tipo, la ubicación y la descripción breve de la situación. El FieldOfficer también describe las posibles respuestas a la situación de emergencia . Una vez completado el formulario, el FieldOfficer envía el formulario, y en ese momento , el despachador se notifica .
4 . El Dispatcher revisa la información presentada y crea un incidente en el base de datos mediante la invocación del caso de uso OpenIncident . El Dispatcher selecciona una respuesta y reconoce el informe de emergencia .

ConnectionDown (incluir la relación)

1. El FieldOfficer y el Dispatcher se les notifica que se interrumpe la conexión. Ellos son informados de las posibles razones por las que se produciría tal evento (por ejemplo, "¿Es la estación FieldOfficer en un túnel?").
2. La situación se registra en el sistema y se recupera cuando se restablezca la conexión.
3. El FieldOfficer y el Dispatcher entran en contacto a través de otros medios, y el Dispatcher inicia ReportEmergency de la estación de Dispatcher.
El caso de uso se extiende ConnectionDown cualquier caso de uso en el que la comunicación entre el FieldOfficer y el despachador puede perderse.
ConnectionDown (extender relación)
1. El FieldOfficer y el Dispatcher se les notifica que se interrumpe la conexión. Ellos son informados de las posibles razones por las que se produciría tal evento (por ejemplo, "¿Es la estación FieldOfficer en un túnel?").
2. La situación se registra en el sistema y se recupera cuando se restablezca la conexión.
3. El FieldOfficer y el contacto enterin Dispatcher a través de otros medios y el ispatcher inicia ReportEmergency de la estación de Dispatcher.

Obtención de requerimientos

Mucha heurística Se ha propuesto en la literatura para la identificación de objetos. Aquí hay unos pocos seleccionados:

Durante la obtención de requisitos, los objetos que participan se generan para cada caso de uso. Si dos casos de uso se refieren al mismo concepto, el objeto correspondiente debe ser el mismo. Si dos objetos comparten el mismo nombre y no se corresponden con el mismo concepto, uno o ambos conceptos se cambiaran el nombre para reconocer y destacar su diferencia. Esta consolidación elimina cualquier ambigüedad en la terminología

utilizada. Por ejemplo, la Tabla 4-2 muestra los objetos que participan iniciales identificadas en el caso de uso ReportEmergency.
Heurística para la identificación de objetos iniciales de análisis:

• Los términos que los desarrolladores o usuarios deben aclarar para entender el caso de uso
• nombres recurrentes en los casos de uso (por ejemplo, de incidentes).
• Las entidades del mundo real que el sistema debe realizar un seguimiento (por ejemplo, FieldOfficer, recursos).
• Real- mundo procesos que el sistema debe realizar un seguimiento (por ejemplo, EmergencyOperationsPlan).
• Los casos de uso (por ejemplo, ReportEmergency).
• Las fuentes de datos o sumideros (por ejemplo, impresoras).
• Los artefactos con los que interactúa el usuario (por ejemplo, la Estación)
• Utilice siempre los términos de dominio de aplicación.

Tabla 4-2 Participación objetos para el caso de uso ReportEmergency.

Dispatcher: El oficial de policía que maneja incidentes. Se abre una Dispatcher, documentos, y se cierra en respuesta a incidentes EmergencyReports y otras comunicaciones con FieldOfficers. Los despachadores son identificados por números de placa.

EmergencyReport: Informe inicial sobre un Incidente desde un FieldOfficer a un Dispatcher. Un EmergencyReport generalmente desencadena la creación de un incidente por el Dispatcher. Un EmergencyReport se compone de un nivel de emergencia, un tipo (incendio, accidente de tráfico, otros), una ubicación y una descripción.

FieldOfficer: Policía o oficial de bomberos de guardia. A FieldOfficer se puede asignar a un máximo de Un incidente en un momento. FieldOfficers se identifican por números de placa.

Situación: de incidentes que requieran la atención de un FieldOfficer. Un incidente puede ser
reportado en el sistema por un FieldOfficer o cualquier otra persona externa al sistema. Un

incidente se compone de una descripción, una respuesta, un estado (abierto, cerrado, documentado), una ubicación y un número de FieldOfficers.

Una vez que los objetos participantes se identifican y consolidan, los desarrolladores pueden utilizar como una lista de control para asegurarse de que el conjunto de casos de uso identificados se ha completado.

Heurística para casos de uso y objetos participantes

- ¿Qué caso de uso crea este objeto (es decir, durante que caso de uso se crean los valores de los atributos del objeto entrados en el sistema)?
- ¿Qué actores pueden acceder a esta información?
- ¿Qué caso modifica y destruye este objeto (es decir, que caso de uso edita o remueve esta información del sistema)?
- ¿Qué actor puede iniciar estos casos de uso?
- ¿Es este objeto necesario (es decir, hay al menos algún caso de uso que dependa de esta información)?

Identificación de los requerimientos no funcionales

Los requisitos no funcionales describen aspectos del sistema que no están directamente relacionados con la conducta funcional. Los requisitos no funcionales abarcan una serie de cuestiones que van desde la interfaz de usuario y los requisitos de tiempo de respuesta a los problemas de seguridad. Requisitos no funcionales se también como requerimientos funcionales porque tienen tanto impacto en el desarrollo y el costo del sistema.

Por ejemplo, considere una pantalla de mosaico que un controlador aéreo utiliza para rastrear aviones. Un sistema de visualización compila los datos de una serie de radares y bases de datos (de ahí el término "mosaico") en una pantalla de resumen que indica todas las aeronaves en un área determinada, incluyendo su identificación, la velocidad y la altitud. El número de aeronaves de un sistema de este tipo puede presentar limitaciones el rendimiento del controlador de tráfico aéreo y el costo del sistema. Si el sistema sólo puede manejar unos pocos aviones de forma simultánea, el sistema no puede ser utilizado en los aeropuertos ocupados. Por otro lado, un sistema capaz de manejar un gran número de aeronaves es más costoso y más complejo a construir y probar.

Los requerimientos no funcionales pueden afectar el trabajo del usuario en formas inesperadas. A fin de obtener con precisión todos los requerimientos no funcionales esenciales, tanto el cliente y el desarrollador deben colaborar para que se identifiquen

(mínimamente), cuales son los atributos del sistema que son difíciles de realizar son fundamentales para el trabajo del usuario. En el ejemplo de pantalla, el número de aeronaves que una sola pantalla debe ser capaz de manejar tiene implicaciones sobre el tamaño de los iconos que se utilizan para la visualización de la aeronave, las características para la identificación de las aeronaves y sus propiedades, la frecuencia de actualización de los datos, y así sucesivamente.

El resultado conjunto de requisitos no funcionales típicamente incluye conflicto de requerimientos. Por ejemplo, los requisitos no funcionales del SatWatch (Figura 4-3) se llaman mecanismo exacto FORAN, de modo que el tiempo nunca necesita ser restablecido, y a un bajo costo unitario, de manera que es aceptable para el usuario reemplazar el reloj con un uno nuevo cuando se rompe.

Estos dos requerimientos no funcionales tienen conflicto en que el precio unitario del reloj incrementa con su exactitud. Para hacer frente a este tipo de conflictos, el cliente y el desarrollador priorizar los requisitos no funcionales, para que puedan ser tratados consistentemente durante la realización del sistema.

Tabla 4-3 Ejemplos de preguntas para la obtención de requerimientos no funcionales.

Categoría	Ejemplo de preguntas
Usabilidad	¿Cuál es el nivel de experiencia del usuario? ¿Qué estándares de interfaz de usuario son familiares para el usuario? ¿Qué documentación debe proporcionarse a los usuarios?
Confiabilidad (incluyendo robustez, seguridad, y la seguridad)	¿Qué tan confiable, disponible y robusta debe ser el sistema? ¿Reiniciar el sistema es aceptable en caso de una falla? ¿Cuántos datos puede perder el sistema? ¿Cómo debe manejar el sistema de excepciones? ¿Existen requerimientos de seguridad del sistema?
Rendimiento	¿Qué tan sensible debe ser el sistema? ¿Hay alguna tarea del usuario que sea crítica? ¿Cuántos usuarios concurrentes debería soportar? ¿Qué tan grande es un almacén de datos típica para sistemas similares? ¿Qué es lo peor latencia que sea aceptable para los usuarios?

Compatibilidad (incluyendo mantenibilidad y portabilidad)	¿Cuáles son las extensiones previstas en el sistema? ¿Quién mantiene el sistema? ¿Existen planes para portar el sistema para diferentes programas o entornos de hardware?
Implementación	¿Existen limitaciones en la plataforma de hardware? ¿Están limitaciones impuestas por el equipo de mantenimiento? ¿Están limitaciones impuestas por el equipo de pruebas?
Interfaz	¿El sistema debe interactuar con cualquier otro sistema existente? ¿Cómo son los datos exportados / importados en el sistema? ¿Qué normas de uso por parte del cliente debe ser soportado por el sistema?
Operación	¿Quién administra el sistema que ejecuta?
Empaque(Packaing)	¿Quién instala el sistema? ¿Cuántas instalaciones se han previsto? ¿Existen limitaciones de tiempo en la instalación?
Legal	¿Hay alguna cuestión de responsabilidad asociados con fallas en el sistema? ¿Hay alguna regalía o derechos de licencia generados por la utilización específica de algoritmos o componentes?

Por desgracia, hay pocos métodos sistemáticos para la obtención de los requisitos no funcionales. En la práctica, los analistas utilizan una taxonomía de los requisitos no funcionales (por ejemplo, los FURPS + esquema, descrito anteriormente) para generar listas de verificación de preguntas para ayudar a que el cliente y los desarrolladores a concentrarse en los aspectos no funcionales del sistema. A medida que los actores del sistema ya han sido identificados en este momento, esta lista de verificación puede ser organizado por el rol y distribuido a usuarios representativos. La ventaja de este tipo de listas de comprobación es que pueden ser reutilizados y expandidas por cada nuevo sistema en un dominio de aplicación dado, reduciendo así el número de omisiones. Tenga en cuenta que tales listas de verificación también pueden dar lugar a la obtención de los requerimientos funcionales adicionales. Por ejemplo, cuando se hacen preguntas sobre el funcionamiento del sistema, el cliente y los desarrolladores pueden descubrir una serie de casos de uso relacionados con la administración del sistema. La tabla 4-3 muestra ejemplos de preguntas para cada una de la categoría FURPS.

Una vez que el cliente y los desarrolladores identifican una serie de requisitos no funcionales, pueden organizarlos en refinamiento y gráficos de dependencia para

determinar las necesidades más funcionales e identificar los conflictos. Si desea más información sobre este tema, se remite al lector a la literatura especializada (por ejemplo, [Chung et al., 1999]).

Gestión de la obtención de requerimientos

En la sección anterior, describimos las cuestiones técnicas de modelado de un sistema en términos de casos de uso. Usar modelado de casos sin embargo, no constituye la obtención de requerimientos. Incluso después de que se conviertan en expertos modeladores de casos de uso, los desarrolladores aun necesitan obtener los requerimientos de los usuarios y llegar a un acuerdo con el cliente. En esta sección, se describen los métodos para la obtención de la información de los usuarios y la negociación de un acuerdo con un cliente. En particular, se describe:

- Negociación de las especificaciones con el cliente: Conjunto de diseño de aplicaciones.
- El mantenimiento de la trazabilidad.
- Documentación de obtención de requerimientos.

Negociación de las especificaciones con el cliente: Conjunto de diseño de aplicaciones.

Conjunto de diseño de aplicaciones (Joint Application Design JAD) es un método de requerimientos desarrollado en IBM a final de los 70s. Su eficacia radica en que el trabajo la obtención de requerimientos se hace en una sola sesión del taller en el que todos actores participan. Los usuarios, clientes, desarrolladores y un líder de la sesión se sientan juntos en una habitación para presentar sus puntos de vista, escuchar otros puntos de vista, negociar y llegar a una solución mutuamente aceptable. El resultado del taller, el documento final JAD, es un documento de especificación de requisitos completa que incluye definiciones de elementos de datos, flujos de trabajo y las pantallas de la interfaz. Debido a que el documento final es desarrollado por los actores (que son los participantes que no solo tienen interés en el éxito del proyecto, sino que también toman decisiones importantes), el documento final JAD representa un acuerdo entre los usuarios, clientes y desarrolladores y minimiza el cambio en los requerimientos durante el proceso de desarrollo. JAP se compone de cinco actividades (Figura 4-15)

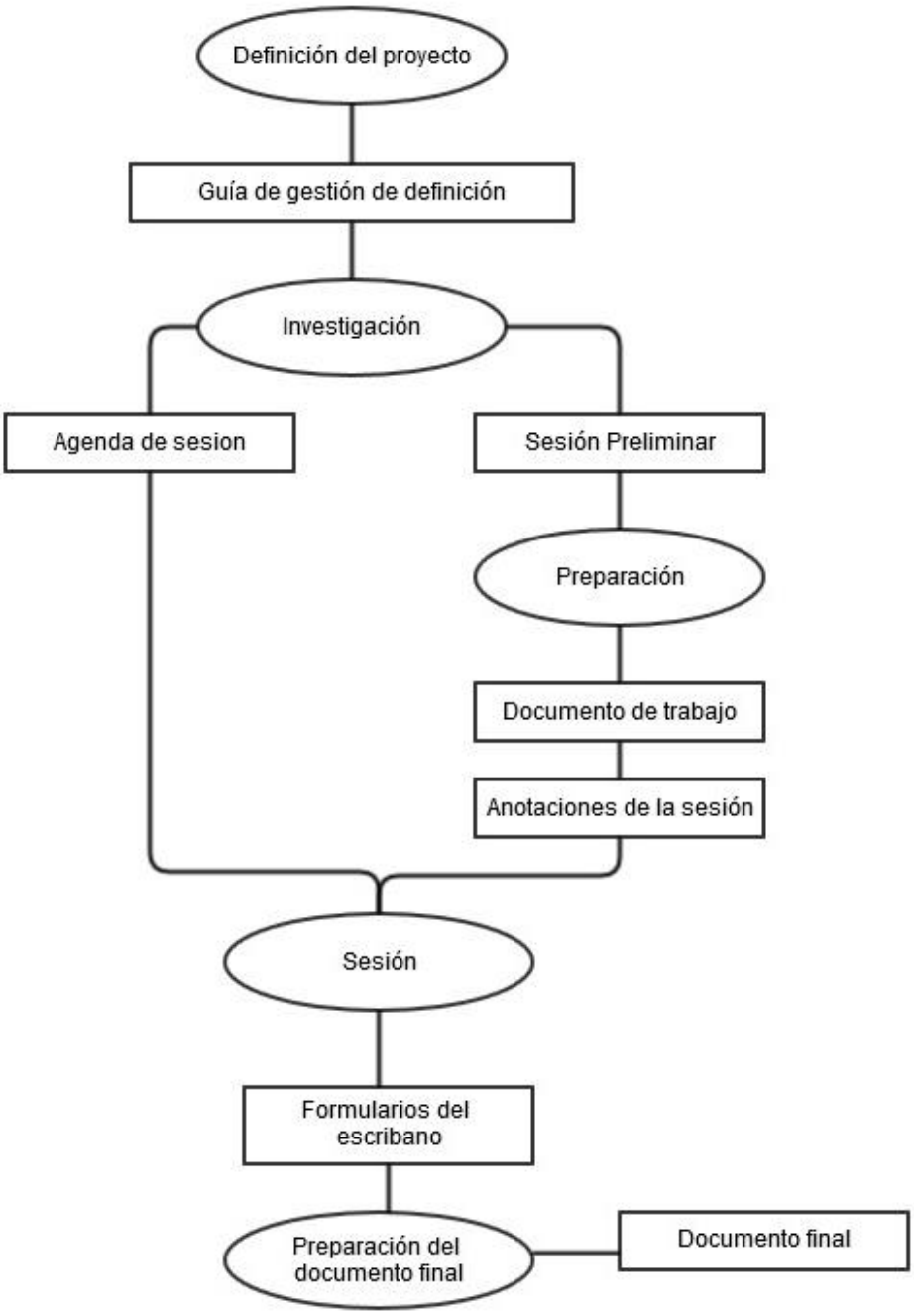


Figura 4-15 Actividades de JAD (diagrama de actividades UML). La parte importante de JAD es la actividad de sesiones durante la cual los actores diseñan y acuerdan la especificación de requerimientos. Las actividades previas a la Sesión maximizan su eficiencia. La producción del documento final captura las decisiones tomadas durante la sesión.

1. *Definición del proyecto*. Durante esta actividad, el facilitador JAD entrevista al jefe del proyecto y al cliente para determinar los objetivos y el alcance del proyecto. Los resultados de las entrevistas se recogen en la Guía de gestión de definición.
2. *Investigación*. Durante esta actividad, el facilitador JAD entrevista a usuarios presentes y finales, reúne información sobre el dominio de aplicación, y describe un primer conjunto de casos de uso de alto nivel. El facilitador JAD también comienza una lista de problemas que se abordarán durante la sesión. Los resultados de esta actividad son una sesión de agenda y un listado de la especificación preliminar e información del sistema.
3. *Preparación*. Durante esta actividad, el facilitador JAD prepara la sesión. El facilitador JAD crea un documento de trabajo, que es el primer borrador del documento final, una agenda para la sesión, y diapositivas para o rotafolios que presentando la información recopilada durante la actividad de investigación. El facilitador JAD también selecciona un equipo compuesto por el cliente, el director del proyecto, los usuarios seleccionados, y los desarrolladores. Todos los actores están representados, y los participantes son capaces de tomar decisiones vinculantes.
4. *Sesión*. Durante esta actividad, el facilitador JAD guía al equipo en la creación de la especificación de requerimientos. Una sesión de JAD tiene una duración de 3 a 5 días. El equipo define y acuerda en los escenarios, casos de uso, y la interfaz de usuario. Todas las decisiones son documentadas por un escribano.
5. *Documento final*. El facilitador JAD prepara el Documento Final, revisando el documento de trabajo para incluir todas las decisiones tomadas durante la sesión. En el documento final representa una especificación completa del sistema acordado durante la sesión. El Documento Final se distribuye a los participantes en la sesión de revisión. Luego, los participantes asisten a una de reunión de 1-2 horas para discutir los comentarios y finalizar el documento.

JAD ha sido utilizada por IBM y otras compañías. JAD aprovecha la dinámica de grupo para mejorar la comunicación entre los participantes y para acelerar la conclusión. Al final de una sesión de JAD, los desarrolladores están más informados de las necesidades del usuario, y los usuarios tienen más conocimientos de desarrollo de compensaciones. Otros beneficios son la reducción de las actividades de rediseño de posteriores Debido a su dependencia de la dinámica social, el éxito de una sesión de JAD a menudo depende de las calificaciones del facilitador. Para una descripción detallada de la JAD, el lector es referido a [Wood & Silver, 1989].

Mantenimiento de la trazabilidad

La trazabilidad es la habilidad de seguir la vida de un requerimiento. Esto incluye conocer de dónde vinieron los requerimientos (por ejemplo: quien los originó, la necesidad del cliente) a cuales aspectos del sistema y del proyecto afectan (por ejemplo cuales componentes realizan el requerimiento, cual caso de prueba checa la realización). Trazabilidad permite a los desarrolladores mostrar que el sistema está completo, a los probadores (testers) les permite mostrar que el sistema cumple con los requerimientos, a los diseñadores les permite registrar el fundamento del sistema, y los mantenedores les permite evaluar el impacto del cambio.

Considere el sistema SatWatch que introdujimos al comienzo del capítulo. En la actualidad, la especificación exige una pantalla de dos líneas que incluye el tiempo y la fecha. Después de que el cliente decide que el tamaño de las cifras es demasiado pequeño para una lectura cómoda, los desarrolladores cambiar la pantalla requisito de una pantalla de una sola línea en combinación con un botón para cambiar entre el tiempo y la fecha. La trazabilidad nos permitirá responder a las siguientes preguntas:

- ¿Quién originó el requisito pantalla de dos líneas?
- ¿Alguna de limitaciones implícitas exigen este requisito?
- ¿Qué componentes debe ser cambiado por el botón adicional y la pantalla?
- ¿Cuáles casos de prueba que deben cambiar?

El método más sencillo para el mantenimiento de la trazabilidad es el uso de referencias cruzadas entre documentos, modelos y artefactos de código. Cada elemento individual (por ejemplo, requisito, componente, clase, operación, caso de prueba) se identifica por un número único. Las dependencias se documentan de forma manual como una referencia cruzada textual que contiene el número del elemento de origen y el número del elemento de destino. Una herramienta de apoyo puede ser tan simple como una hoja de cálculo o una herramienta de procesamiento de textos. Este enfoque es costoso en tiempo y en personal, y es propenso a errores. Sin embargo, para los pequeños proyectos, los desarrolladores pueden observar beneficios.

Para proyectos de gran escala, herramientas de bases de datos especializadas permiten la automatización parcial de la captura, edición y vinculación de las dependencias de trazabilidad a un nivel más detallado (por ejemplo, DOORS [Telelogic] o RequisitePro [rational]). Estas herramientas reducen el costo de mantenimiento de la trazabilidad, pero requieren la compra y la formación de la mayoría de los actores e imponen restricciones a otras herramientas en el proceso de desarrollo.

Documentación de obtención de requerimientos.

Los resultados de la obtención requerimientos y las actividades de análisis se documentan en el documento Análisis de Requerimientos (RAD). Este documento describe completamente el sistema en términos de requisitos funcionales y no funcionales. La audiencia para el RAD incluye el cliente, los usuarios, la gestión de proyectos, los analistas de sistemas (es decir, los desarrolladores que participan en los requisitos), y los diseñadores del sistema (es decir, los desarrolladores que participan en el diseño del sistema). La primera parte del documento, que incluye los casos de uso y requisitos no funcionales, se escribe durante la obtención de requerimientos. La formalización de la especificación en términos de modelos de objetos se escribe durante el análisis. Figura 4-16 es una plantilla de ejemplo para un RAD utilizada en este libro.

La primera sección de la RAD es una introducción. Su propósito es proporcionar una breve descripción de la función del sistema y de las razones de su desarrollo, su ámbito de aplicación, y las referencias al contexto de desarrollo (por ejemplo, la referencia a la declaración del problema escrito por el cliente, las referencias a los sistemas existentes, estudios de viabilidad). La introducción también incluye los objetivos y criterios de éxito del proyecto.

En la segunda sección, el sistema actual, describe el estado actual de las cosas. Si el nuevo sistema reemplazará un sistema existente, esta sección se describe las funciones y los problemas

---

Documento de análisis de requerimientos

1. Introducción
   1.1 Objetivo del sistema
   1.2 Ámbito de aplicación del sistema
   1.3 Objetivos y criterios de éxito del proyecto
   1.4 Definiciones, acrónimos y abreviaturas
   1.5 Referencias
   1.6 Información general
2. sistema actual
3. sistema propuesto

3.1 Información general

3.2 Requisitos funcionales

3.3 Los requisitos no funcionales

3.3.1 Usabilidad

3.3.2 Fiabilidad

3.3.3 Rendimiento

3.3.4 Compatibilidad

3.3.5 Implementación

3.3.6 Interfaz

3.3.7 Packaging

3.3.8 Legal

3.4 Los modelos de sistema

3.4.1 Escenarios

3.4.2 Modelo de casos de uso

3.4.3 Modelo de objetos

3.4.4 Modelo dinámico

3.4.5 Interfaz del usuario-rutas de navegación y pantallas

4. Glosario

Figura 4-16 Esquema del Documento de Análisis de Requerimientos (RAD). Secciones en cursiva se completan durante el análisis (véase el capítulo siguiente).

del sistema actual. De lo contrario, esta sección describe como las tareas son soportadas por el nuevo sistema. Por ejemplo, en el caso de SatWatch, el usuario restablece el reloj cada que viaja a otro zona horaria. Debido a la naturaleza de esta operación el usuario ocasionalmente establece la hora incorrecta y ocasionalmente se niega a restablecerla. En contraste, SatWatch va a asegurar continuamente que se tenga la hora correcta. En el caso de FRIEND, el actual sistema basado en papel: los despachadores mantiene rastro de la

asignación de recursos por medio de formularios. La comunicación entre los despachadores y personal de campo es por radio. El sistema actual requiere un alto costo de la documentación y de gestión que FRIEND tiene como objetivo reducir. En la tercera sección, el sistema propuesto, documenta la obtención los requerimientos y el modelo de análisis del nuevo sistema. Se divide en cuatro secciones:

ARENA. Planteamiento del problema.

1.- Problema

La popularidad de la Internet y la World Wide Web ha permitido la creación de una variedad de comunidades virtuales, grupos de personas que compartiendo intereses en común, pero nunca se han conocido en persona.

Estas comunidades virtuales pueden ser de corta duración (por ejemplo, un grupo de personas que se reúnen en una sala de chat o jugar un torneo) o de larga vida (por ejemplo, los suscriptores de una lista de correo). Pueden incluir un pequeño grupo de personas o miles.

Muchos juegos de ordenador multijugador incluyen ahora soporte para las comunidades virtuales que son los jugadores del juego dado. Los jugadores pueden recibir noticias acerca de las actualizaciones del juego, nuevos mapas del juego y los personajes, ya que pueden anunciar y organizar partidos, comparar las calificaciones y intercambiar consejos. La compañía de juegos se aprovecha de esta infraestructura para generar ingresos o para la publicidad de sus productos.

En la actualidad, sin embargo, cada compañía de juegos se desarrolla este tipo de apoyo de la comunidad en cada juego individual. Cada empresa utiliza una infraestructura diferente, diferentes conceptos, y proporciona diferentes niveles de apoyo.

Esta redundancia y los resultados de inconsistencia en muchas desventajas, incluyendo una curva de aprendizaje para los jugadores cuando se unen cada nueva comunidad, para las compañías de juegos que necesitan para desarrollar el apoyo desde el principio, y para los anunciantes que necesitan para ponerse en contacto con cada comunidad por separado. Por otra parte, esta solución no ofrece mucha oportunidad para que la fertilización cruzada entre las diferentes comunidades.

2.- Objetivos.

Los objetivos del proyecto ARENA son los siguientes:

- Proporcionar una infraestructura para el funcionamiento de un estadio, incluyendo el registro de nuevos juegos y jugadores, la organización de torneos, y hacer el seguimiento de las puntuaciones de los jugadores.
- Proporcionar un marco para los dueños de la liga para personalizar el número y el orden de partidos y la acumulación de expertos puntos de rating.
- Proporcionar un marco para los desarrolladores de juegos para el desarrollo de nuevos juegos, o para la adaptación de los juegos existentes en el marco de ARENA.
- Proporcionar una infraestructura para los anunciantes.

3.- Requerimientos funcionales.

ARENA soporta cinco tipos de usuarios:

- El operador debe ser capaz de definir nuevos juegos, definir nuevos estilos de torneo (por ejemplo, ronda los torneos, campeonatos, lo mejor de la serie), definir nuevas fórmulas de calificación de expertos, y gestionar los usuarios.
- Los dueños de la liga deben ser capaces de definir una nueva liga, organizar y anunciar nuevos torneos dentro de una liga, llevar a cabo un torneo, y declarar un ganador.
- Los jugadores deben ser capaces de registrar en una arena, solicitar una liga, jugar los partidos que se asignan al jugador, o dejar fuera del torneo.
- Los espectadores deben ser capaces de controlar un partido en curso y comprobar las puntuaciones y estadísticas de partidos, y los jugadores del pasado. Los espectadores no tienen que registrarse en una arena.
- El anunciante debe poder cargar nuevos anuncios, seleccione un esquema de anuncio (por ejemplo, patrocinador del torneo, el patrocinador de la liga), revisar el balance de vencimiento y cancelará anuncios.

4.- Requerimientos no funcionales.

- Bajo costo operativo. El operador debe ser capaz de instalar y administrar una arena sin necesidad de adquirir componentes de software adicionales y sin la ayuda de un administrador de sistemas a tiempo completo.
- Extensibilidad. El operador debe ser capaz de añadir nuevos juegos, nuevos estilos de torneos y nuevas fórmulas de calificación de expertos. Tales adiciones pueden requerir que el sistema sea cerrado temporalmente y nuevos módulos (por ejemplo, clases de Java) que se añade al sistema. Sin embargo, ninguna modificación del sistema existente debe ser requerido.

- Escalabilidad. El sistema debe ser compatible con el pistoletazo de salida de muchos torneos paralelos (por ejemplo, 10), cada uno que implican un máximo de 64 jugadores y varios cientos de espectadores simultáneos.
- Red de bajo ancho de banda. Los jugadores deben ser capaces de jugar partidos a través de un módem analógico de 56K o más rápido.

5.- Entorno de destino.

- Todos los usuarios deben ser capaces de acceder a cualquier escenario con un navegador con soporte para applets cookies, JavaScript y Java. Las funciones de administración (por ejemplo, añadiendo nuevos juegos, estilos del torneo, y los usuarios) utilizados por el operador no debe estar disponible a través de la web.
- ARENA debería funcionar en cualquier sistema operativo Unix (por ejemplo, MacOS X, Linux, Solaris).

Identificacion de actores y escenarios.

Identificamos cinco actores , uno para cada tipo de usuario en el enunciado del problema (Operador , LeagueOwner , Jugador , Espectador, y Advertiser) . A medida que la funcionalidad básica del sistema es la de organizar y jugar torneos , primero desarrollamos un escenario de ejemplo , organizar - TicTacToeTournament (Figura 4-18) para obtener y explorar esta funcionalidad en más detalle. Por primera se centra en un corte vertical estrecho del sistema , comprendemos mejor las expectativas del cliente del sistema , incluyendo los límites del sistema y el tipo de interacción entre el usuario y el sistema. Utilizando el escenario organizeTicTacToeTournament de Figura 4-18 , producimos una serie de preguntas para que el cliente se muestra en la (Figura 4-19) . Con base en las respuestas del cliente , refinamos el escenario en consecuencia.

Tenga en cuenta que cuando se hacen preguntas de un cliente, nuestro principal objetivo es entender las necesidades del cliente y el dominio de aplicación . Una vez que entendemos el dominio y producir una primera versión de la especificación de requisitos , podemos empezar a comparar las características y el costo de los requisitos del cliente y priorizando . Sin embargo , entrelazando elicitación y negociación demasiado temprano es generalmente contraproducente .

Después de que refinamos el primer escenario hasta el punto de que ambos estamos de acuerdo con el cliente en el límite del sistema (para ese escenario) , nos centramos en el ámbito de aplicación general del sistema . Esto se realiza mediante la identificación de una serie de escenarios más cortos para cada actor. Inicialmente, estos escenarios no se detallan , pero en cambio, abarcan una amplia gama de funciones (Figura 4-20) .

Escenarios específicos Cuando encontramos discrepancias o ambigüedades , que con más detalle. En este ejemplo , los escenarios defineKnockOutStyle y installTicTacToeGame se refinan a un nivel comparable de detalle que el organizeTicTacToeTournament (Figura 4-18) .

Nombre del escenario:
organizeTicTacToeTournament
Participating actor instances alice:Operator, joe:LeagueOwner, bill:Spectator, mary:Player

Flujo de eventos.

1. Joe , un amigo de Alice, es un aficionado TicTacToe y voluntarios para organizar un torneo.
2. Alice registra Joe en la arena como propietario de una liga.
3. Joe define primero una liga TicTacToe principiantes , en el que cualquier jugador pueden ser admitidos . Esta liga , dedicado a los juegos de Tic Tac Toe , estipula que los torneos disputados en esta liga seguirán el estilo de torneo eliminatorio y " Winner Takes All" fórmula .
4. Joe programa el primer torneo de la liga de 16 jugadores a partir del día siguiente.
5. Joe anuncia el torneo en una variedad de foros en la Web y envía correo a otros miembros de la comunidad TicTacToe .
6. Bill y Mary reciben la notificación por correo electrónico .
7. María está interesado en jugar el torneo y registros. Aplican otras 19 personas.
8. Joe registra 16 jugadores para el torneo y rechaza los 4 que aplica pasado.
9. Los 16 jugadores, incluyendo a María , reciben un token electrónico para entrar en el torneo y el momento de su primer partido .
10. Otros suscriptores de la lista de correo TicTacToe , incluyendo a Bill , reciben un segundo aviso sobre el torneo , incluyendo el nombre de los jugadores y el calendario de partidos .
11. Como Joe comienza el torneo, los jugadores tienen una cantidad limitada de tiempo para entrar en el partido . Si un jugador no se presenta , pierde el juego.
12. María juega su primer partido y gana. Ella avanza en el torneo y está programado para el próximo partido contra el otro ganador de la primera ronda .

13. Después de visitar la página principal del torneo TicTacToe , Bill cuenta de la victoria de María y decide vigilarla próximo partido. Él selecciona el partido, y ve la secuencia de movimientos de cada jugador a medida que ocurren . También ve un banner de publicidad en la parte inferior de su navegador , haciendo publicidad de otros torneos y productos de tic tac toe .
14. El torneo continúa hasta el último partido, momento en el que se declara el ganador del torneo y su récord de la liga se le atribuye todos los puntos relacionados con el torneo.
15. Además, el ganador del torneo se acumula puntos de rating expertos.
16. Joe puede optar por programar más torneos de la liga, en cuyo caso , los jugadores conocidos son notificados de la fecha y la prioridad que se da a través de los nuevos jugadores.

Los escenarios típicos, una vez refinado, abarcan varias páginas de texto. También empezamos a mantener un glosario de términos importantes, para garantizar la coherencia en la especificación y para asegurarnos de que utilizamos términos del cliente. Rápidamente nos damos cuenta de que los términos partido, juego, torneo, y Liga representan conceptos del dominio de aplicación que deben ser definidos con precisión, ya que estos términos pueden tener una interpretación diferente en otros contextos de juego. Para ello, mantenemos un glosario de trabajo y revisamos nuestras definiciones como nuestro trabajo exploratorio progresa (Tabla 4-4).

ARENA. Caso de estudio.

Pasos 2, 7: Los diferentes actores se registra en el sistema. En el primer caso, el administrador registra a Joe como propietario de una liga, en el segundo caso, un jugador sí se registra a si mismo en el sistema.

- Registro de usuarios debe seguir el mismo paradigma. ¿Quién proporciona la información de registro y cómo se revisa la información, validada y aceptada?
- Cliente: Dos procesos se confunden en los pasos 2 y 7, el proceso de registro, en el que los nuevos usuarios (por ejemplo, un jugador o un dueño de la liga) establecen su identidad, y el proceso de solicitud, en el que los jugadores indican que quieren tomar parte en un torneo específico. Durante el proceso de registro, el usuario proporcione información personal (nombre, apodo, E-mail) y sus intereses (los tipos de juegos y torneos que quieren estar informados sobre). La información es validada por el operador. Durante el proceso de solicitud, los jugadores indican que el torneo quieren participar pulg Esto es utilizado por el dueño de la liga durante la programación del partido.
- Dado que la información del jugador ya ha sido validada por el operador, se debe programar el partido automatizarse por completo?
- Cliente: Sí, por supuesto.

Paso 5: Joe envía correo a los miembros de la comunidad TicTacToe:

- ¿Proporciona ARENA la oportunidad de los usuarios suscribirse a listas de correo individuales?
- Cliente: Sí. Deben existir listas de correo para anunciar nuevos juegos, nuevas ligas, nuevos torneos, etc
- ¿Tiene tienda ARENA un perfil de usuario (por ejemplo, juego observaba, partidos jugados, los intereses que especifique un estudio de usuarios) a los efectos de la publicidad?
- Cliente: Sí, pero los usuarios deben ser capaces de registrar sin completar una encuesta a los usuarios, si así lo desean. Ellos deben ser alentados a entrar en el estudio, pero esto no debe impedir su entrada. Serán expuestos a los anuncios de todos modos.
- En caso de que el perfil se utilizará para suscribirse automáticamente a las listas de correo?

- Cliente: No, creemos que los usuarios de nuestra comunidad preferirían tener un control completo sobre sus suscripciones a listas de correo. Suscripciones de adivinanzas no les darían la impresión de que están en control.

Paso 13: Bill navega estadísticas del partido y decide ver el próximo partido en tiempo real.

- ¿Cómo se identifican los jugadores a los espectadores? Por nombre real, por E-mail, por sobrenombre?
- Cliente: Esto debería dejarse en manos del usuario durante el registro.
- ¿Puede un espectador reproducir partidos viejos?
- Cliente: Los juegos deben ser capaces de proporcionar esta capacidad, pero algunos juegos (por ejemplo, en tiempo real, juegos de acción 3D) pueden optar por no hacerlo debido a las limitaciones de recursos.
- ARENA debe apoyar los juegos en tiempo real?
- Cliente: Sí, estos representan la mayor parte de nuestro mercado. En general, ARENA debe apoyar la más amplia gama de juegos como sea posible.

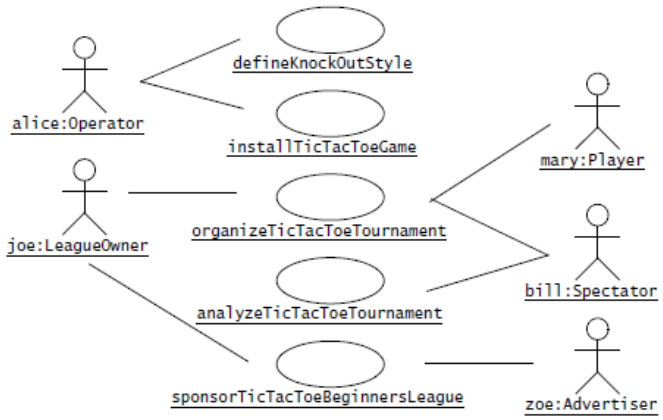


Figura 4-20 Escenarios de alto nivel identificados por ARENA. Los clientes y los desarrolladores inicialmente describen brevemente los escenarios. Ellos los perfeccionan aún más para aclarar ambigüedades o descubrir los desacuerdos.

Tabla 4-4. Glosario Trabajado por ARENA. Hacer un seguimiento de los términos más importantes y sus definiciones asegura la consistencia en la memoria y se asegura de que los desarrolladores utilicen el idioma del cliente.

Juego	El juego es una competencia entre un número de jugadores que se llevó a cabo de acuerdo con un conjunto de reglas. En ARENA, el término del juego se refiere a una pieza de software que hace cumplir el conjunto de normas, realiza un seguimiento del progreso de cada jugador, y decide el ganador. Por ejemplo, tictactoe y el ajedrez son juegos.
Partido	Un partido es una contienda entre dos o más jugadores siguiendo las reglas de un juego. El resultado de un partido puede ser un solo ganador y un juego de perdedores o un empate (en la que su hay ganadores ni perdedores). Algunos juegos pueden no permitir lazos.
Torneo	Un torneo es una serie de partidos entre un conjunto de jugadores. Torneos terminan con un solo ganador. La forma en que los jugadores acumulan puntos y partidos están programados por la Liga en la que el torneo está organizado.
Liga	A League representa una comunidad para el funcionamiento de Torneos. A la Liga se asocia con un juego específico y TournamentStyle. Los jugadores registrados en la Sociedad se acumulan puntos según el ExpertRating definido en la Liga. Por ejemplo, un juego de ajedrez de la Liga novato tiene una fórmula ExpertRating diferente a un experto de la Liga.

Estilo de torneo	El TournamentStyle define el número de partidos, y su secuencia para un determinado conjunto de jugadores. Por ejemplo, los jugadores se enfrentan a todos los otros jugadores en el torneo de una sola vez en un TournamentStyle round robin.

Una vez que estamos de acuerdo con el cliente en un ámbito general del sistema, formalizamos los conocimientos adquiridos hasta el momento en forma de casos de uso de alto nivel.

4.6.3 Identificación de Casos de Uso

Generalizar escenarios en los casos de uso permite a los desarrolladores estar un paso atrás de las situaciones concretas y considerar el caso general . Los desarrolladores pueden consolidar funcionalidad relacionada en casos de uso individuales y dividir la funcionalidad relacionada en varios casos de uso.

Al inspeccionar el escenario organizeTicTacToeTournament de cerca, nos damos cuenta de que cubre una amplia gama de funcionalidad iniciado por muchos actores. Anticipamos que la generalización de este escenario se traduciría en un caso de uso de varias decenas de páginas, y tratar de dividirlo en casos independientes y de uso independiente iniciados por actores individuales. En primer lugar, decidimos dividir la funcionalidad relacionada con las cuentas de usuario en dos casos de uso, ManageUserAccounts , iniciadas por el operador , y del Registro , iniciada por los jugadores potenciales y dueños de la liga (Figura 4-21) . Identificamos un nuevo actor, Anonymous , representando estos potenciales usuarios que aún no tienen una cuenta . Del mismo modo, nos dividimos la funcionalidad con la navegación partidos pasados y con la gestión de perfiles de usuario en los casos de uso independientes (BrowseTournamentHistory y ManageOwnProfile , iniciada por el espectador y el jugador , respectivamente). Por último , para acortar aún más el uso OrganizeTournament caso , dividimos fuera de la funcionalidad para la creación de nuevas ligas en el caso de uso DefineLeague , como un LeagueOwner puede crear muchos torneos dentro del alcance de una sola liga . Por el contrario , prevemos que la instalación de nuevos juegos y nuevos estilos requiere pasos similares del operador . Por lo tanto , consolidamos todas las funciones relacionadas con la instalación de nuevos componentes en el caso ManageComponents uso iniciada por el operador .

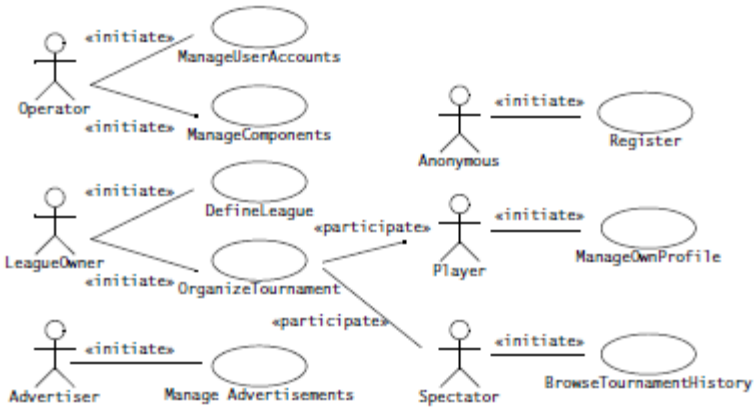
Capturamos estas decisiones dibujando un diagrama de casos de visión general y describiendo brevemente cada caso de uso (Figura 4-21) . Tenga en cuenta que un diagrama de casos de uso por sí solo no describe tanto la funcionalidad. En lugar de ello , es un índice dentro de las muchas descripciones producidos durante esta fase .

A continuación, se describen los campos de cada caso de uso de alto nivel , entre ellos los actores participantes , las condiciones de entrada y salida , y un flujo de eventos. La figura 4-22 muestra el caso de uso OrganizeTournament de alto nivel.

Tenga en cuenta que todos los pasos de este flujo de eventos describen las acciones de actores . Casos de uso de alto nivel se centran principalmente en la tarea realizada por el actor. La interacción detallada con el sistema, y las decisiones sobre los límites del sistema , se pospuso inicialmente a la fase de refinamiento.

Esto nos permite describir primero el dominio de aplicación con los casos de uso , la captura , en particular, cómo los distintos actores colaboran para lograr sus objetivos.

En la Figura 4-22 , se describe la secuencia de acciones que se realizan por cuatro actores para organizar un torneo : el LeagueOwner , que facilita la actividad completa , del Anunciante, para resolver los problemas de patrocinio exclusivos, los jugadores potenciales que quieran participar , y los espectadores . En el primer paso , se describe el manejo de la cuestión de patrocinio , lo que hace evidente que cualquier problema de patrocinio debe ser resuelto antes de que el torneo se anuncian, y antes de que los jugadores se aplican para el torneo. Originalmente , el tema del patrocinio no se describió claramente en los escenarios de la Figura 4-20 (que sólo describió los patrocinios de las ligas) .



Register	Los usuarios anónimos se registran con un espacio para un jugador o una cuenta de Liga-propietario. Se requieren cuentas de usuario antes de solicitar un torneo o la organización de una liga. Los espectadores no necesitan cuentas.
ManageUserAccounts	El Operador acepta registros de propietarios de la liga y de los jugadores, cancela las cuentas existentes, e interactúa con los usuarios acerca de la ampliación de sus cuentas.
ManageComponents	El operador instala nuevos juegos y define nuevos estilos de

	torneos (generaliza defineKnockOutStyle y installTicTacToeGame).
DefineLeague	El LeagueOwner define una nueva liga (generaliza los primeros pasos de la organizeTicTacToeTournament escenario).
OrganizaTournament	La propietario de la liga crea y anuncia un nuevo torneo, acepta aplicaciones de reproducción, los horarios de los partidos, y comienza el torneo. Durante el torneo, los jugadores juegan partidos y los espectadores siguen los partidos. Al final del torneo, los jugadores se acreditan con puntos (generaliza la organizeTicTacToeTournament escenario).
ManageAdvertisements	El anunciante sube banners y patrocinadores de la liga o torneos (generaliza sponsorTicTacToeBeginnersLeague).
ManageOwnProfile	Los jugadores a administran sus suscripciones a listas de correo y contestan una encuesta de marketing.
BrowseTournamentHistory	Los Espectadores examinan las estadísticas de los torneos y estadísticas de los jugadores y partidos de reproducción que ya han sido concluidos (generaliza la analyzeTicTacToeTournament escenario).

Nombre del caso de uso	OrganizeTournament
Actores participantes	Iniciado por Propietario de liga Se comunica con el anunciante, Player, y Espectador
Flujo de eventos	1. El LeagueOwner crea un Torneo, solicita patrocinios de Anunciantes, y anuncia el Torneo (incluye casos de uso AnnounceTournament). 2. Los jugadores se aplican para el Torneo (incluya casos de uso ApplyForTournament). 3. El LeagueOwner procesa las solicitudes del jugador y los asigna a los partidos (incluya ProcessApplications de casos de uso). 4. El LeagueOwner comienza el Torneo (incluye casos de uso KickoffTournament). 5. Los jugadores compiten en los torneos como estaba previsto

	y los espectadores ven los partidos (incluir casos de uso PlayMatch). 6. El LeagueOwner declara el ganador y archiva el torneo (incluye casos de uso ArchiveTournament).
Condición de entrada	El LeagueOwner se registra en ARENA.
Condiciones de salida	El LeagueOwner registra un nuevo torneo en el registro de ARENA y el ganador ha acumulado nuevos puntos en la liga, Ó The LeagueOwner cancela el torneo y la posición de los jugadores en la liga no se ha modificado.

Figura 4-22 Un ejemplo de un caso de uso de alto nivel, OrganizeTournament.

Después de las conversaciones con el cliente, decidimos manejar también el patrocinio del torneo, y para manejar al principio de cada torneo. Por un lado, esto permite que nuevos patrocinadores que se añadan al sistema, y por otro lado, permite al patrocinador, a cambio, anunciar el torneo utilizando sus propios recursos. Por último, esto permite que el sistema para seleccionar mejor banners de publicidad durante el proceso de solicitud.

En este caso de uso de alto nivel, que reducía los elementos esenciales del escenario organizar-TicTacToeTournament en seis pasos y dejamos los detalles para el caso de uso detallado. Mediante la descripción de cada caso de uso de alto nivel de este modo, captamos todas las relaciones entre los actores que el sistema debe tener en cuenta. Esto también se traduce en una descripción sucinta del sistema que sea comprensible para cualquier recién llegado al proyecto.

A continuación, escribimos los casos de uso detalladas para especificar las interacciones entre los actores y el sistema.

4.6.4 Refinación de casos de uso e identificación de relaciones.

La refinación de los Casos de uso permite a los desarrolladores definir con precisión la información intercambiada entre los actores y el sistema. La refinación de los Casos de uso también permite el descubrimiento de los flujos alternativos de eventos y excepciones que el sistema debe manejar.

Para mantener el estudio de casos manejable, no mostramos un entorno refinado. Empezamos por identificar un caso de uso detallado de cada paso del flujo de los acontecimientos en el caso de uso OrganizeTournament de alto nivel. El diagrama de casos de uso resultante se muestra en la Figura 4-23. Luego nos centramos en el caso de uso, AnnounceTournament: Figura 4-24 contiene una descripción del flujo de los acontecimientos, y la Figura 4-25 se identifican las excepciones que podrían ocurrir en AnnounceTournament. Los casos de uso restantes se desarrollarán de manera similar.

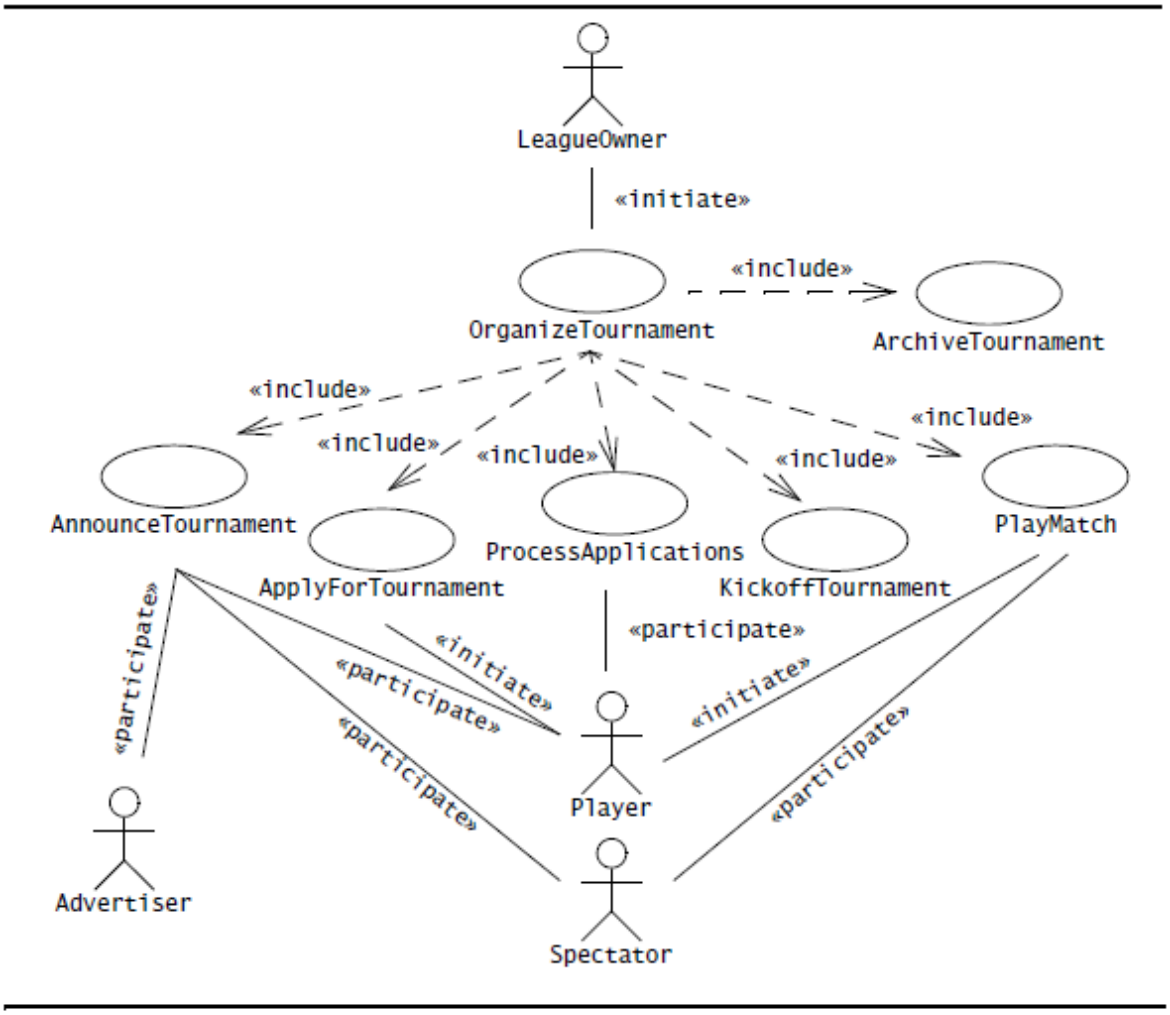


Figure 4-23 Detailed use cases refining the `OrganizeTournament` high-level use case.

Todos los casos de uso de la figura 4-23 son iniciados por el LeagueOwner, excepto que el ApplyForTournament y PlayMatch son iniciadas por el jugador. El Anunciante participa en AnnounceTournament y el espectador participa en los casos de uso AnnounceTournament y uso PlayMatch.

Nombre	AnnounceTournament
Actores que participan	Iniciado por LeagueOwner Se comunica con el jugador, el anunciante y el espectador.
Flujo de eventos	1 . El LeagueOwner solicita la creación de un torneo. 2 . El sistema comprueba si el LeagueOwner ha superado el número de torneos en la liga o en la arena. Si no , el sistema presenta la LeagueOwner con una forma . 3 . El LeagueOwner especifica un nombre, inicio de aplicaciones y fin fechas durante las cuales los jugadores pueden aplicar para el torneo, fechas de inicio y finalización para la realización del torneo , y un número máximo de jugadores . 4 . El sistema solicita al LeagueOwner si un patrocinio exclusivo debe ser buscada y , en caso afirmativo, se presenta una lista de anunciantes que expresaron el deseo de ser patrocinadores exclusivos. 5 . Si el LeagueOwner decide buscar un patrocinador exclusivo , selecciona un subconjunto de los nombres de los patrocinadores propuestos. 6 . El sistema notifica a los patrocinadores seleccionados sobre el próximo torneo y la tarifa plana para patrocinios exclusivos. 7 . El sistema se comunica sus respuestas a la LeagueOwner . 8 . Si hay patrocinadores interesados , el LeagueOwner selecciona uno de ellos . 9 . El sistema registra el nombre del patrocinador exclusivo y carga la tarifa plana para patrocinios a la cuenta del anunciante . A partir de ahora , todos los banners de publicidad relacionados con el torneo son proporcionados por sólo el patrocinador exclusivo . 10.Otherwise , si no se seleccionó ningún patrocinadores (ya sea porque no Anunciante estaba interesado o la LeagueOwner no ha seleccionado uno) , los banners publicitarios son seleccionados al azar y se carga a la cuenta del anunciante en una base unitaria . 11.Una vez que el tema del patrocinio está cerrado, el sistema solicita al LeagueOwner con una lista de grupos de jugadores, espectadores y anunciantes que podrían estar interesados en el nuevo torneo . 12.El LeagueOwner selecciona qué grupos de notificar .

	13.El sistema crea una página de inicio en el campo para el torneo. Esta página se utiliza como punto de entrada para el torneo (por ejemplo , para proveer a los jugadores interesados un formulario para solicitar el torneo, y para interesar a los espectadores en la observación de los partidos) . 14.En la fecha de inicio de la aplicación , el sistema notifica a cada usuario interesado mediante el envío de un enlace a la página principal del torneo. Los jugadores pueden luego solicitar el torneo con el caso de uso ApplyForTournament hasta la fecha de finalización de la aplicación.

Figure 4-24 Un ejemplo de un caso de uso a detalle, AnnounceTournament.

Condiciones de entrada	El LeagueOwner inicia sesión en ARENA
Condiciones de salida	• El patrocinio del torneo se resuelva: o bien un solo anunciante exclusivo pagó una tarifa fija o banners se extraerá al azar de la piscina común de publicidad en la Arena. • Posibles jugadores recibieron un aviso relativo a la próxima torneo y pueden solicitar su participación. • Potenciales espectadores recibieron un aviso relativo a la próxima torneo y saben cuando el torneo está a punto de comenzar. • La página de inicio de torneo está disponible para cualquier de ver, por lo tanto, otros espectadores potenciales pueden encontrar la página de inicio del torneo a través de motores de búsqueda web, o navegar por la página principal Arena.
Requisitos de calidad	• Ofertas y respuestas de anunciantes requieren autenticación segura, por lo que los anunciantes pueden facturar únicamente en sus respuestas. • Los anunciantes deben ser capaces de cancelar los contratos de patrocinio en un plazo fijo, como es requerido por las leyes locales.

Figure 4-24 Continuación

relaciones también. Empezamos por escribir el flujo de los acontecimientos para el caso de uso AnnounceTournament (Figura 4-24).

Los pasos de la Figura 4-24 describen en detalle la información que se intercambia entre el actor y el sistema. Nótese, sin embargo, que no nos describimos los detalles de la interfaz de usuario (por ejemplo, formas, botones, diseño de ventanas o páginas web). Es mucho más fácil diseñar una interfaz de usuario usable después, luego de que sabemos la intención y la responsabilidad de cada actor. Por lo tanto, el enfoque en la fase de refinamiento es asignar (o descubrir) la intención detallada y responsabilidades de cada actor.

Al describir los pasos del caso de uso detallado AnnounceTournament, nosotros y el cliente hizo más decisiones sobre los límites del sistema:

Hemos introducido las fechas de inicio y finalización del proceso de solicitud y para ejecutar el torneo (paso 3 en la Figura 4-24). Esto nos permite comunicarnos plazos a todos los actores involucrados para asegurar que el torneo ocurre dentro de un plazo razonable.

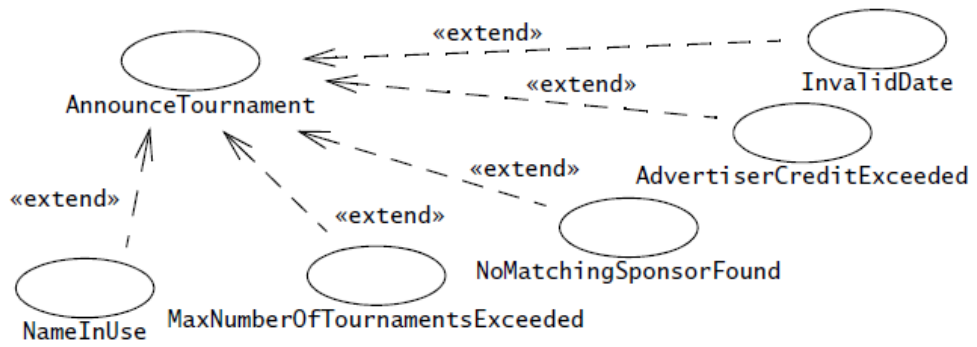
Decidimos que los anunciantes indican en su perfil si están interesados en patrocinios exclusivos o no. Esto permite que el LeagueOwner para apuntar Publicidad más específicamente (Paso 4 en la figura 4-24).

También decidimos permitir que los anunciantes se comprometan a acuerdos de patrocinio a través del sistema automatizado y la contabilidad de la publicidad y de la facturación. Esto implica requisitos legales y de seguridad en el sistema, que documentamos en el campo "requisitos de calidad" del caso de uso (Paso 9 y 10 en la Figura 4-24).

Tenga en cuenta que estas decisiones se validan con el cliente. Diferentes clientes y entornos puede conducir a la evaluación de las compensaciones de forma diferente para el mismo sistema. Por ejemplo, la decisión acerca de solicitar anunciantes obtiene un compromiso a través de los resultados del sistema que resulta en un sistema complejo y caro. Una alternativa habría sido para solicitar anunciantes a través de E-mail, pero obtener su compromiso a través del teléfono. Esto habría dado lugar a un sistema más sencillo, pero más trabajo en la parte de la DueñoDeLaLiga. El cliente es la persona que decide entre tales alternativas, comprendiendo, por supuesto, que estas decisiones tienen un impacto en el costo y la fecha de entrega del sistema.

A continuación, se identifican las excepciones que pueden presentarse durante el caso de uso detallado. Esto se hace mediante la revisión de cada paso en el caso de uso y la identificación de todos los eventos que podrían salir mal. Nosotros describimos brevemente el manejo de cada excepción y representamos los casos de uso de manejo de excepciones como extensiones del caso de uso AnuncioDeTorneo (Figura 4-25) .

Tenga en cuenta que no todas las excepciones son iguales, y diferentes tipos de excepciones se abordan mejor en diferentes etapas de desarrollo . En la Figura 4-25, identificamos excepciones provocadas por las restricciones de recursos(MaxNumberOfTournamentsExceeded), la entrada del usuario no válido (InvalidDate, NameInUse), o restricciones del dominio de aplicación (AvertiserCreditExceeded, NoMatchingSponsorFound).
Las excepciones asociadas a las limitaciones de recursos se manejan mejor en el diseño del sistema. Sólo durante el diseño del sistema se convertirá en claro que los recursos son limitados y la mejor forma es compartir los entre los



AvertiserCreditExceeded	El sistema elimina el anunciante de la lista de posibles

	patrocinadores.
InvalidDate	El sistema informa al LeagueOwner y solicita una nueva fecha.
MaxNumberOfTournamentsExceeded	El caso de uso AnnounceTournament se termina.
NameInUse	El sistema informa al LeagueOwner y pide un nuevo nombre.
NoMatchingSponsorFound	El sistema se salta los pasos exclusivos patrocinador y elige al azar anuncios de la publicidad de la piscina.

Figura 4-25 Las excepciones ocurren en AnunciarTorneo representados como una ampliación de los casos de uso. (Nota que AnunciarTorneo en esta figura es el mismo que el caso de uso en la figura 4-23).

distintos usuarios que pueden , a su vez , dar lugar a nuevas actividades de requisitos durante el diseño del sistema para validar con el cliente el manejo de tales excepciones. Las excepciones asociadas con la entrada del usuario no válidas se manejan mejor en el diseño de la interfaz de usuario, cuando los desarrolladores serán capaz de decidir en qué punto para comprobar si hay una entrada no válida, cómo mostrar mensajes de error, y cómo
evitar entradas no válidas en el primer lugar. La tercera categoría de excepciones – restricciones de dominio de la aplicación- deben recibir la atención del cliente y del desarrollador temprano. Estas son las excepciones que por lo general no son evidentes para el desarrollador. Cuando son desapercibidas, requieren retrabajo sustancial y cambios en el sistema. Una manera sistemática para obtener esas excepciones es ir a través del caso uso paso a paso el caso con el cliente o un experto en el área.

Muchos acontecimientos excepcionales pueden ser representados ya sea como una excepción (por ejemplo, AdvertiserCreditExceeded) o como un requisito no funcional (por ejemplo , "un anunciante no debe ser capaz de gastar más dinero del límite fijo del anuncio un acordado de antemano con el Operador durante el registro") . Esta última representación es más apropiada para el las restricciones globales que se aplican a varios casos de uso. A la inversa , el primero es más apropiado para eventos que solo pueden ocurrir en un caso de uso (por ejemplo, "NoMatchingSponsorFound ") .

Escribir cada caso de uso detallado, incluyendo sus excepciones, constituye la parte del esfuerzo del león en la obtención de requisitos. Lo ideal es que los desarrolladores escriban cada caso de uso detallado y dirigan todas las dudas del dominio de la aplicación antes de comprometerse con el proyecto y el inicio de la realización del sistema. En la práctica, esto nunca sucede. Para los grandes sistemas, los desarrolladores producen una gran cantidad de documentación en la que es difícil, si no imposible, mantener la consistencia. Peor aún, la actividad de obtención de requerimientos de grandes proyectos ya debería financiarse, ya que esta fase requiere una gran cantidad de recursos tanto en el cliente y la organización de desarrollo.

Por otra parte, la integridad en una etapa temprana puede ser contraproducente : los pasos del en el caso de uso cambian mientras se descubren nuevos hechos de dominio durante el desarrollo. La decisión sobre la cantidad de casos de uso para detallar y la cantidad que se debe dejar implícita es tanto una cuestión de confianza como de la economía: el cliente y los desarrolladores deben compartir una suficientemente buena comprensión de que el sistema estará listo para comprometerse a un horario, un presupuesto y un proceso para manejar los cambios futuros (incluidos cambios en los requisitos, el programa y presupuesto).

En ARENA , nos centramos en la especificación a detalle de las interacciones que involucran a los anunciantes y los jugadores, ya que tienen un papel fundamental en la generación de ingresos. Casos de uso relacionados con la administración del sistema o la instalación de nuevos juegos o estilos del torneo se dejan para más tarde, ya que también incluyen cuestiones más técnicas que dependen de la solución de dominio.

4.6.5 Identificación de requisitos no funcionales

Los requisitos no funcionales provienen de una variedad de fuentes durante la obtención. El problema que empezamos con la figura 4-17 ya ha especificado el rendimiento y los requisitos de implementación. Al detallar el caso de uso AnnounceTournament, identificamos requisitos de orden jurídico para los anunciantes de facturación. Al revisar las excepciones en la sección anterior, identificamos una restricción de la cantidad de dinero que los anunciantes pueden gastar. Aunque nos encontramos con muchos de los requisitos no funcionales al escribir casos de uso y el perfeccionamiento de ellos, no podemos garantizar que identifiquemos todos los requisitos no funcionales esenciales. Para asegurar que se ha completado, se utilizan las categorías del FURPS + que describimos en la Sección 4.3.2 (o cualquier otra taxonomía sistemática de requisitos no funcionales) como una lista de control para hacer preguntas del cliente. La Tabla 4-5 representa los requisitos no funcionales que identificamos en ARENA después de detallar el caso de uso de AnnounceTournament.

Tabla 4-5 **Requisitos no funcionales** consolidados para ARENA, después de la primera versión de la detallada
del caso de uso de AnnounceTournament.

Categoría	**Requisitos no funcionales**
Usabilidad	• Los espectadores deben poder acceder a los juegos en progreso sin previo registro y sin conocimiento previo del juego.
Confiabilidad	• Los accidentes debidos a errores de software en componentes del juego deben interrumpir en máximo un torneo con el juego. El resto de los Torneos en progreso debe continuar normalmente. •Cuando un torneo es interrumpido debido a un accidente, su LeagueOwner debe ser capaz de reiniciar el Torneo. A lo sumo, sólo el último movimiento de cada partido interrumpido se puede perder.
Rendimiento	• El sistema debe ser compatible con el pistoletazo de salida de muchos torneos paralelos (por ejemplo, 10), involucrando hasta 64 jugadores y varios cientos de espectadores simultáneos. • Los jugadores deben ser capaces de jugar partidos a través de un módem analógico.
Compatibilidad	• El operador debe ser capaz de añadir nuevos juegos y nuevos TournamentStyles. Tales adiciones pueden requieren que el sistema sea cerrado temporalmente y nuevos módulos (por ejemplo, clases de Java) se añadan para el sistema. Sin embargo, ninguna modificación del sistema existente debe ser requerida.
Implementación	• Todos los usuarios deben ser capaces de acceder a un estadio con un navegador web applets, cookies de apoyo, JavaScript y Java. La administración funciones utilizadas por el operador no están disponibles a través de la web. • ARENA debería funcionar en cualquier sistema operativo Unix (por ejemplo, MacOS X, Linux, Solaris).
Operación	•Un anunciante no debería ser capaz de gastar más dinero en publicidad que un límite fijo acordado de antemano con el operador durante el registro.
Legal	• Ofertas y propuestas de anunciantes requieren autenticación segura, por lo que los acuerdos se pueden construir únicamente en sus respuestas. • Los anunciantes deben ser capaces de cancelar los contratos de patrocinio dentro de un período fijo, como es requerido por las leyes locales.

Lecciones aprendidas

En esta sección, desarrollamos un caso de uso inicial y análisis de un modelo de objetos basados en un enunciado proporcionado por el cliente. Usamos escenario y preguntas como herramientas de obtención para aclarar conceptos ambiguos e información faltante. También obtuvimos requerimientos no funcionales. Aprendimos que:

- La obtención de requerimientos involucra el intercambio constante de perspectivas. (alto nivel vs detallado, clientes vs desarrollados, actividad vs entidad).
- La obtención de requerimientos requiere una sustancial participación del cliente.
- Los desarrolladores no deben asumir que saben lo que el cliente quiere.
- La obtención de requerimientos no funcionales fuerza a los actores a hacer y documentar intercambios.

Lecturas adicionales

El concepto de caso de uso se hizo popular por Ivar Jacobson en su excelente libro, Ingeniería de Software Orientada a Objetos: Un enfoque de casos de uso [Jacobson et al, 1992.]. Para un relato de las primeras investigaciones sobre los requisitos basados en escenarios y, más en general, en el diseño participativo. Diseño basado en escenarios [Carroll, 1995] incluye muchos trabajos de los principales investigadores sobre los escenarios y casos de uso. Este libro también describe las limitaciones y dificultades de los requisitos basados en escenarios y diseño participativo, que siguen siendo válidos hoy en día.

Para orientación método específico, software para su uso [Constantine & Lockwood, 1999] contiene mucho material sobre la especificación de los sistemas utilizables con los casos de uso, incluyendo la obtención de conocimiento impreciso de los usuarios y clientes, un tema suave que por lo general no está cubierto de software de ingeniería de los libros de texto. Redacción eficaz los casos de uso [Cockburn, 2001] y su sitio web que acompaña http://www.usecases.org proporcionan muchos heurísticas prácticas para escribir casos de uso textualmente (en lugar de sólo dibujarlos).

Los usuarios finales tienen un papel crítico durante la obtención de requerimientos. Norman ilustra esto mediante el uso de ejemplos de objetos de uso cotidiano, tales como puertas, estufas, grifos y [Norman, 2002]. Argumenta que los usuarios no deben esperar que leer un manual de usuario y aprender nuevas habilidades para cada producto a los que están expuestos. En lugar de ello, el conocimiento sobre el uso del producto, tales como pistas que indican en qué dirección se abre una puerta, debe ser empotrado en su diseño. Toma ejemplos de objetos de uso cotidiano, pero los mismos principios se aplican a los sistemas informáticos y el diseño de interfaces de usuario.

El mundo de la ingeniería de requisitos es mucho más pobre cuando se trata de lidiar con requisitos no funcionales. El Marco de NFR, descrito en requisitos no funcionales en

Ingeniería de Software [Chung et al., 1999], es uno de los pocos métodos que aborda este tema de manera sistemática y en profundidad.

La plantilla RAD introducido en este capítulo es sólo un ejemplo de cómo organizar un documento de requisitos. IEEE publicó la documentación estándar IEEE-Std 830-1998 para especificaciones de requisitos de software [IEEE Std . 830-1998] . El apéndice de la norma contiene varias muestras esboza para la descripción de los requisitos específicos. Los ejemplos de este capítulo siguen un enfoque dialéctico para la obtención de requerimientos, un proceso de discusión y negociación entre los desarrolladores, el cliente y los usuarios finales. Este enfoque funciona bien cuando el cliente es el usuario final, o cuando el cliente tiene un conocimiento suficientemente detallado del dominio de aplicación. En los grandes sistemas, como un sistema de control del tráfico aéreo, no solo usuario o cliente tiene una perspectiva completa del sistema. En estas situaciones, la dialéctica enfoque rompe, tanto conocimiento implícito sobre las actividades de los usuarios no es encontrado hasta que fue demasiado tarde. En la última década, la etnografía, un método de campo de la antropología, ha ganado popularidad en la ingeniería de requisitos. Con este enfoque, los analistas se sumergen sí mismos en el mundo de los usuarios, observe su trabajo diario, y participar en sus reuniones. Los analistas anotan sus observaciones desde un punto de vista neutral. El objetivo de este enfoque es para descubrir el conocimiento implícito. El método de la coherencia, se informa en el análisis social en el proceso de ingeniería de requerimientos: desde la etnografía al método [Viller y Sommerville, 1999], proporciona un ejemplo práctico de la etnografía aplicada a la ingeniería de requisitos.

La gestión de la trazabilidad más allá de los requisitos es todavía un tema de investigación, se remite al lector a la literatura especializada [Jarke, 1998]. Por último, Requisitos de software y especificaciones: un léxico de la práctica, los principios y los prejuicios [Jackson, 1995] es una pieza breve, incisivo y entretenido que ofrece muchos conocimientos sobre los principios y métodos de la ingeniería de requisitos.

Ejercicios

1. Considere su reloj como un sistema y establecer el tiempo de 2 minutos por delante. Anote cada interacción entre usted y su reloj como un escenario. Anote todas las interacciones, incluyendo cualquier comentario que el reloj que ofrece.
2. Considere la situación que escribiste en el Ejercicio 4-1. Identifique el actor del escenario. A continuación, escriba el correspondiente caso de uso (SetTime). Incluya todos los casos, e incluyen ajuste de la hora hacia adelante y hacia atrás, y el establecimiento de horas, minutos y segundos.
3. Suponga que el sistema de vigilancia que se describe en los ejercicios 4-1 y 4-2 también es compatible con una función de alarma. Describir ajusta la alarma como un caso independiente llamado uso SetAlarmTime.

4. Examine los casos Settime y uso SetAlarmTime que escribiste en los ejercicios 4-2 y 4-3. Eliminar las redundancias utilizando una relación de incluir. Justificar por qué una relación incluir es preferible a una relación de extensión en este caso.
5. Suponga que el FieldOfficer puede invocar una función de ayuda al llenar un EmergencyReport. La característica HelpReportEmergency proporciona una descripción detallada de cada campo y especifica qué campos son obligatorios. Modifique el caso de uso ReportEmergency (descrito en la Figura 4-10) para incluir esta funcionalidad ayuda. ¿Qué relación debe utilizar para relacionar el ReportEmergency y HelpReportEmergency?
6. A continuación se presentan ejemplos de los requisitos no funcionales. Indique cuáles de estos requisitos son verificables y que no lo son:
    - "El sistema debe ser utilizable."
    - "El sistema debe tener retroalimentación en un segundo de la emisión de un comando"
    - "La disponibilidad del sistema debe estar por encima de 95 por ciento."
    - "La interfaz de usuario del nuevo sistema debe ser lo suficientemente similar al antiguo sistema que los usuarios familiarizados con el viejo sistema pueden ser fácilmente entrenados para usar el nuevo sistema".
7. La necesidad de desarrollar una especificación completa puede alentar a un analista para escribir documentos detallados y prolongados. ¿Qué calidad de especificación (véase Tabla 4-1) puede animar a un analista para mantener la especificación corta?
8. El mantenimiento de la trazabilidad en requerimientos y las actividades subsiguientes es caro, debido a la información adicional que debe ser capturado y mantenido. ¿Cuáles son los beneficios de la trazabilidad que superan esta sobrecarga? ¿Cuál de esos beneficios beneficia directamente al analista?
9. Explique por qué los cuestionarios de opción múltiple, como el medio principal para la extracción de información del usuario, no son efectivos para producir requisitos.
10. Desde su punto de vista, describir las fortalezas y debilidades de los usuarios durante la actividad de la obtención de requerimientos. Describe también las fortalezas y debilidades de desarrolladores durante la actividad de la obtención de requerimientos.
11. Defina brevemente el término "menú". Escribe tu respuesta en un pedazo de papel y ponlo en la mesa junto con las definiciones de otros cuatro estudiantes. Comparar las cinco definiciones y discutir cualquier diferencia sustancial.
12. Escriba el ManageAdvertisement de casos de uso de alto nivel iniciado por el anunciante y escribir casos de uso detalladas refinar este caso de uso de alto nivel. Considere características que permiten a un anunciante para subir banners de

publicidad, para asociar palabras clave con cada banner, para suscribirse a avisos de nuevos torneos de ligas o juegos específicos, y para controlar los gastos y pagos realizados en la cuenta de publicidad. Asegúrese de que sus casos de uso también son consistentes con el planteamiento del problema ARENA presenta en la Figura 4-17.

13. Teniendo en cuenta el caso de uso AnnounceTournament en la Figura 4-24 escribe el flujo del evento, las condiciones de entrada, y las condiciones de salida para el caso ApplyForTournament uso, iniciado por un jugador interesado en participar en el torneo de nueva creación. Considere también la declaración del problema ARENA presenta en la Figura 4-17. Escriba una lista de preguntas para el cliente cuando se encuentra con cualquier otra alternativa.
14. Escriba los flujos de eventos, las condiciones de entrada y condiciones de salida para el caso de uso excepcional para AnnounceTournament muestran en la Figura 4-25. Usa relaciones si es necesario eliminar la redundancia.