Ingeniería de Software I

Casos de Uso

Un Método Práctico para Explorar Requerimientos

Santiago Ceria

1. Introducción

1.1. Objetivo de este apunte

En uno de los párrafos más citados del artículo por lejos más citado en la bibliografía de la Ingeniería del Software, Frederick P. Brooks [Brooks87], dice: *"La parte más difícil de construir un sistema es precisamente saber qué construir. Ninguna otra parte del trabajo conceptual es tan difícil como establecer los requerimientos técnicos detallados, incluyendo todas las interfaces con gente, máquinas, y otros sistemas. Ninguna otra parte del trabajo afecta tanto al sistema si es hecha mal. Ninguna es tan difícil de corregir mas adelante... Entonces, la tarea más importante que el ingeniero de software hace para el cliente es la extracción iterativa y el refinamiento de los requerimientos del producto"*.

Los casos de uso son un método que, justamente, ayudan al Ingeniero de Software a llevar adelante esta parte del desarrollo de un sistema de software.

Si bien sus antecedentes tienen ya más de 15 años de antigüedad, la técnica de análisis con caso de uso es relativamente nueva. La bibliografía es bastante escasa y, en muchos casos, tiene pocos consejos prácticos para ayudar al personal de desarrollo de sistemas que intenta aplicarla.

El objetivo principal de este apunte es ayudar a los alumnos de Ingeniería del Software I a aplicar la técnica de análisis con casos de uso en sus trabajos prácticos. Además, esperamos que sea de utilidad para quien quiera aplicarla en un proyecto de desarrollo real.

1.2. ¿Qué son los Casos de Uso?

Los casos de uso son una técnica para especificar el comportamiento de un sistema:

"Un caso de uso es una secuencia de interacciones entre un sistema y alguien o algo que usa alguno de sus servicios."

Todo sistema de software ofrece a su entorno –aquellos que lo usan– una serie de servicios. Un caso de uso es una forma de expresar cómo alguien o algo externo a un sistema lo usa. Cuando decimos "alguien o algo" hacemos referencia a que los sistemas son usados no sólo por personas, sino también por otros sistemas de hardware y software.

Por ejemplo, un sistema de ventas, si pretende tener éxito, debe ofrecer un servicio para ingresar un nuevo pedido de un cliente. Cuando un usuario accede a este servicio, podemos decir que está "ejecutando" el caso de uso *ingresando pedido*.

1.3. Historia

Los Casos de Uso fueron introducidos por Jacobson en 1992 [Jacobson92]. Sin embargo, la idea de especificar un sistema a partir de su interacción con el entorno es original de Mc Menamin y Palmer, dos precursores del análisis estructurado, que escribieron en 1984 un excelente libro cuya lectura recomendamos [McMenamin84]. En ese libro, se define un concepto muy parecido al del caso de uso: el *evento*. Para Mc Menamin y Palmer, un evento es algo que ocurre fuera de los límites del sistema, ante lo cual el sistema debe responder. Siguiendo con nuestro ejemplo anterior, nuestro sistema de ventas tendrá un evento "Cliente hace Pedido". En este caso el sistema deberá responder al estimulo que recibe –el pedido– procesándolo.

Sin embargo, existen algunas diferencias entre los casos de uso y los eventos. Las principales son:

1) Los eventos se centran en describir qué hace el sistema cuando el evento ocurre, mientras que los casos de uso se centran en describir cómo es el diálogo entre el usuario y el sistema.

2) Los eventos son "atómicos": se recibe una entrada, se la procesa, y se genera una salida, mientras que los casos de uso se prolongan a lo largo del tiempo mientras dure la interacción del usuario con el sistema. De esta forma, un caso de uso puede agrupar a varios eventos.

3) Para los eventos, lo importante es qué datos ingresan al sistema o salen de él cuando ocurre el evento (estos datos se llaman datos esenciales), mientras que para los casos de uso la importancia del detalle sobre la información que se intercambia es secundaria. Según esta técnica, ya habrá tiempo más adelante en el desarrollo del sistema para ocuparse de este tema.

Los casos de uso combinan el concepto de evento del análisis estructurado con otra técnica de especificación de requerimientos bastante poco difundida: aquella que dice que una buena forma de expresar los requerimientos de un sistema es escribir su manual de usuario antes de construirlo. Esta técnica, si bien ganó pocos adeptos, se basa en un concepto muy interesante: *al definir requerimientos, es importante describir al sistema desde el punto de vista de aquél que lo va a usar, y no desde el punto de vista del que lo va a construir*. De esta forma, es más fácil validar que los requerimientos documentados son los *verdaderos requerimientos* de los usuarios, ya que éstos comprenderán fácilmente la forma en la que están expresados.

1.4. Aclaraciones Importantes

Este apunte no es sólo un resumen de la bibliografía existente, sino que intenta agregar conceptos que surgieron a partir de la aplicación de los casos de uso en la práctica. Por lo tanto, es importante tener en cuenta que lo que se discute en este apunte, si bien está basado en la bibliografía, no necesariamente refleja estrictamente las definiciones formales sobre los casos de uso. Por ejemplo, la técnica de identificar nuevos casos de uso a partir de casos existentes, discutida más adelante, está basada en el análisis estructurado, pero no es mencionada en la bibliografía que describe los casos de uso.

1.5. Los Casos de Uso y UML

A partir de la publicación del libro de Jacobson, gran parte de los más reconocidos especialistas en métodos Orientados a Objetos coincidieron en considerar a los casos de uso como una excelente forma de especificar el comportamiento externo de un sistema. De esta forma, la notación de los casos de uso fue incorporada al lenguaje estándar de modelado UML –Unified Modelling Language– propuesto por Ivar Jacobson, James Rumbaugh y Grady Booch, tres de los precursores de las metodologías de Análisis y Diseño Orientado a Objetos, y avalado por las principales empresas que desarrollan software en el mundo. UML va en camino de convertirse en un estándar para modelado de sistemas de software de amplia difusión.

A pesar de ser considerada una técnica de Análisis Orientado a Objetos, es importante destacar que los casos de uso poco tienen que ver con entender a un sistema como un conjunto de objetos que interactúan, que es la premisa básica del análisis orientado a objetos "*clásico*". En este sentido, el éxito de los casos de uso no hace más que dar la razón al análisis estructurado, que propone que la mejor forma de empezar a entender un sistema es a partir de los servicios o funciones que ofrece a su entorno, independientemente de los objetos que interactúan dentro del sistema para proveerlos.

Como era de esperar, es probable que en el futuro los métodos de análisis y diseño que prevalezcan hayan adoptado las principales ventajas de todos los métodos disponibles en la actualidad –estructurados, métodos formales, métodos orientados a objetos, etc.–

De lo dicho anteriormente podemos concluir que los casos de uso son independientes del método de diseño que se utilice, y por lo tanto del método de programación. Luego de documentar los requerimientos de un sistema con casos de uso, se puede diseñar un sistema "estructurado" (manteniendo una separación entre datos y funciones), o un sistema Orientado a Objetos, sin que la técnica sea de mayor o menor utilidad en alguno de los dos casos. Esto da más flexibilidad al método, y probablemente contribuya a su éxito.

2. Definiciones Básicas

2.1. Actores

Un actor es una agrupación uniforme de personas, sistemas o máquinas que interactúan con el sistema que estamos construyendo de la misma forma. Por ejemplo, para una empresa que recibe pedidos en forma telefónica, todos los operadores que reciban pedidos y los ingresen en un sistema de ventas, si pueden hacer las mismas cosas con el sistema, son considerados un único actor: *Empleado de Ventas*.

Los actores son externos al sistema que vamos a desarrollar. Por lo tanto, al identificar actores estamos empezando a delimitar el sistema, y a definir su alcance. Definir el alcance del sistema debe ser el primer objetivo de todo analista, ya que un proyecto sin alcance definido nunca podrá alcanzar sus objetivos.

Es importante tener clara la diferencia entre usuario y actor. Un actor es una clase de rol, mientras que un usuario es una persona que, cuando usa el sistema, asume un rol. De esta forma, un usuario puede acceder al sistema como distintos actores. La forma más simple de entender esto es pensar en perfiles de usuario de un sistema operativo. Una misma persona puede acceder al sistema con distintos perfiles, que le permiten hacer cosas distintas. Los perfiles son en este caso equivalentes a los actores.

Otro sistema que interactúa con el que estamos construyendo también es un actor. Por ejemplo, si nuestro sistema deberá generar asientos contables para ser procesados por el sistema de contabilidad, este último sistema será un actor, que usa los servicios de nuestro sistema.

También puede ocurrir que el actor sea una máquina, en el caso en que el software controle sus movimientos, o sea operado por una máquina. Por ejemplo, si estamos construyendo un sistema para mover el brazo de un robot, el hardware del robot será un actor, asumiendo que dentro de nuestro sistema están las rutinas de bajo nivel que controlan al hardware.

Los actores se representan con dibujos simplificados de personas, llamados en inglés “stick man” (hombres de palo).

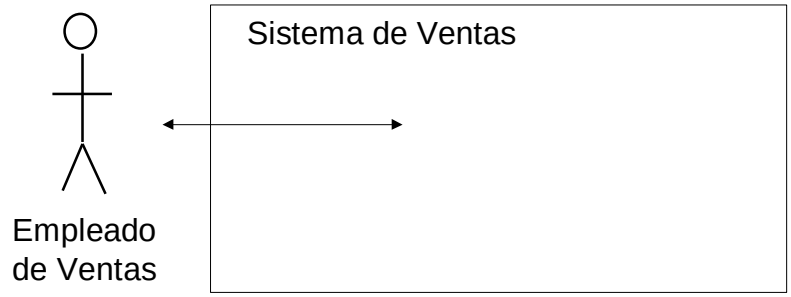

Figura 1 – El empleado de ventas es un actor del sistema de ventas

Si bien en UML los actores siempre se representan con “hombres de palo”, a veces resulta útil representar a otros sistemas con alguna representación más clara.

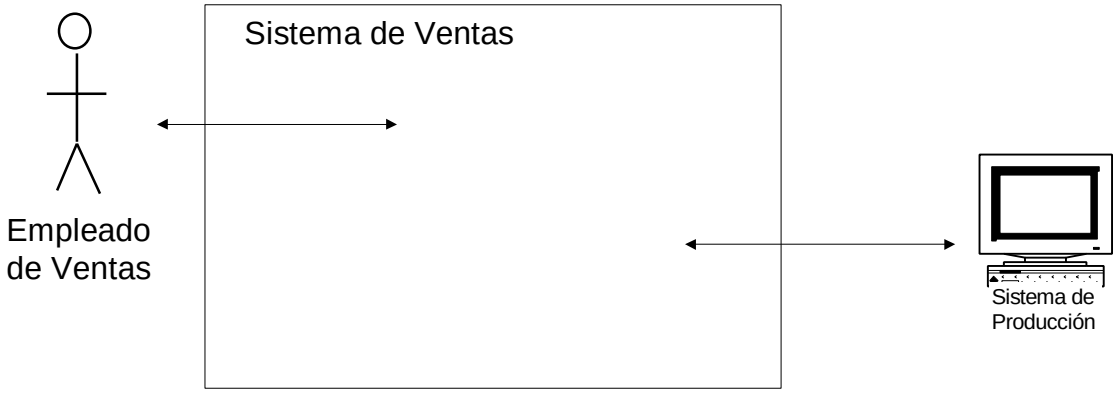

Figura 2 – Cuando el actor es otro sistema se puede cambiar la notación

Las flechas, que existían en la propuesta original de Jacobson pero desaparecieron del modelo semántico de UML, pueden usarse para indicar el flujo de información entre el sistema y el actor. Si la flecha apunta desde el actor hacia el sistema, esto indica que el actor está ingresando información en el sistema. Si la flecha apunta desde el sistema hacia el actor, el sistema está generando información para el actor.

Identificar a los actores es el primer paso para usar la técnica de casos de uso. Por ejemplo, en el sistema de pedidos nombrado anteriormente, sin conocer prácticamente ningún detalle sobre cómo funcionará, podemos decir que:

1) El grupo de usuarios que ingrese pedidos al sistema será un actor.
2) El grupo de usuarios que haga otras operaciones con los pedidos, como por ejemplo autorizarlos, cancelarlos y modificar sus plazos de entrega, será un actor.
3) Todo grupo de usuarios que reciba ciertos informes del sistema, como por ejemplo estadísticas de ventas, será un actor.

Es común que los distintos actores coincidan con distintas áreas de la empresa en la que se implementará el sistema, o con jerarquías dentro de la organización (empleado, supervisor y gerente son distintos actores, si realizan tareas distintas).

Todos los actores participan de los casos de uso. Ahora bien, es lógico que existan intersecciones entre lo que hacen los distintos actores. Por ejemplo, un supervisor puede autorizar pedidos, pero también puede ingresarlos. Veremos más adelante cómo, definiendo actores abstractos, podemos especificar este comportamiento común para evitar redundancia.

2.2. Casos de Uso

Definiciones Básicas

Como mencionamos anteriormente, *un caso de uso es una secuencia de interacciones entre un sistema y alguien o algo que usa alguno de sus servicios.* Un caso de uso es iniciado por un actor. A partir de ese momento, ese actor, junto con otros actores, intercambian datos o control con el sistema, participando de ese caso de uso.

El nombre de un caso de uso se expresa con un verbo en gerundio, seguido generalmente por el principal objeto o entidad del sistema que es afectado por el caso. Gráficamente, los casos de uso se representan con un óvalo, con el nombre del caso en su interior.

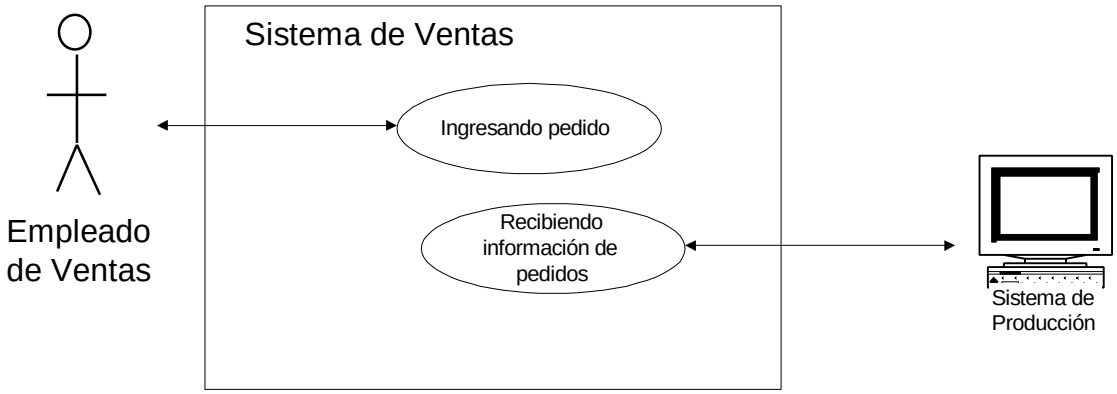

Figura 3 – Los casos de uso se representan gráficamente con óvalos

Es importante notar que el nombre del caso siempre está expresado desde el punto de vista del actor y no desde el punto de vista del sistema. Por eso el segundo caso de uso se llama *Recibiendo información de pedidos* y no *Generando información de pedidos.*

Los casos de uso tienen las siguientes características:

1) Están expresados desde el punto de vista del actor.
2) Se documentan con texto informal.
3) Describen tanto lo que hace el actor como lo que hace el sistema cuando interactúa con él, aunque el énfasis está puesto en la interacción.
4) Son iniciados por un único actor.
5) Están acotados al uso de una determinada funcionalidad –claramente diferenciada– del sistema.

El último punto es tal vez el más difícil de definir. Uno podría, después de todo, decir que todo sistema tiene un único caso de uso *Usando el Sistema.* Sin embargo, la especificación resultante sería de poco utilidad para entenderlo; sería como implementar un gran sistema escribiendo un único programa.

La pregunta importante es: ¿Qué es una "funcionalidad claramente diferenciada"? Por ejemplo, ¿ingresar pedidos es un caso de uso y autorizarlos es otro? ¿Cancelar los pedidos, es otro caso de uso, o es parte del caso de uso referido al ingreso de pedidos? Si bien se pueden encontrar argumentos válidos para cualquiera de las dos alternativas, en principio la respuesta a todas estas preguntas es que son todos casos de uso distintos. Lamentablemente, si en la programación los criterios para dividir la funcionalidad en programas suelen ser difusos, los criterios para dividir la funcionalidad de un sistema en casos de uso son aún más difusos, y por esto se hace importante usar el sentido común en estas decisiones.

En principio podríamos decir que la regla general es: *una función del sistema es un caso de uso si se debe indicar explícitamente al sistema que uno quiere acceder a esa función*. Por ejemplo, si uno quiere dar de alta un pedido, accederá a la funcionalidad de alta de pedidos del sistema. Sin embargo, si uno quiere dar de alta un campo del pedido, no debe indicar al sistema que quiere acceder a esa función. Dar de alta un campo de un pedido es una función que forma parte de un caso de uso mayor: dar de alta un pedido.

Esta regla, si bien puede ser útil, no debe seguirse al pie de la letra, ya que se puede prestar a confusiones. Por ejemplo, supongamos que uno quiere especificar un sistema en el cual los usuarios pueden ver un pedido, y tienen disponibles funciones para ver el siguiente pedido, el anterior, el último y el primero. El actor debe indicar al sistema que quiere acceder a cada una de esas funciones, y según nuestra regla serían todas ellas casos de uso distintos. Sin embargo, en esta situación es mucho más práctico definir un único caso de uso *navegando pedidos,* que especificarlos todos como casos de uso distintos.

Cuando pensamos en el grado de detalle de la división de los casos de uso también resulta útil imaginar que uno está escribiendo el manual del usuario del sistema. A nadie se le ocurriría escribir un manual de usuario con un solo capítulo en el que se describe toda su funcionalidad. De la misma forma, no se debe escribir una especificación con un solo caso de uso. A pesar de saber que uno se quiere mantener lejos de este extremo, la cantidad de capítulos del manual es variable dependiendo de la persona que lo escriba.

Para dar una idea aproximada del nivel de detalle de la división de los casos de uso en casos menores, también podemos pensar que un trabajo práctico de Ingeniería del Software I debiera tener alrededor de 8 casos de uso primarios.

Descripción de los Casos de Uso

Los casos de uso se documentan con texto informal. En general, se usa una lista numerada de los pasos que sigue el actor para interactuar con el sistema. A continuación se muestra una parte simplificada de la descripción del caso de uso "Ingresando Pedido".

Caso de Uso: Ingresando Pedido.
Actor: Empleado de Ventas.
1) El cliente se comunica con la oficina de ventas, e informa su número de cliente
2) El oficial de ventas ingresa el número de cliente en el sistema
3) El sistema obtiene la información básica sobre el cliente
4) El cliente informa el producto que quiere comprar, indicando la cantidad
5) El sistema obtiene la información sobre el producto solicitado, y confirma su disponibilidad.
6) Se repite el paso 4) hasta que el cliente no informa más productos
7) ...

Figura 4 – Descripción simplificada del caso de uso "Ingresando Pedido"

Al describir los casos de uso aparece una de sus principales limitaciones. Supongamos que queremos describir un sistema en el cual la interacción con el usuario es muy simple: ingresa un conjunto básico de datos, y con esos datos el sistema realiza una gran cantidad de cálculos, aplicando complejas fórmulas. ¿Cómo hago con un caso de uso para especificar el comportamiento interno del sistema, si su comportamiento no es trivial? La respuesta es que los casos de uso son muy limitados para lograr este objetivo, ya que se basan en el uso de texto informal. Por lo tanto, deberemos usar una nueva notación para especificar este comportamiento interno, algo equivalente a los diagramas de flujo de datos del análisis estructurado. En UML se propone usar una notación llamada "Diagrama de Actividad", el moderno heredero del diagrama de flujo, o "flowchart".

Otra de las limitaciones de los casos de uso es que no hay una sintaxis clara para indicar, dentro de la descripción del caso, las decisiones e iteraciones. De esta forma, es común que en las descripciones de los casos se deba recurrir a frases como “Se repite el paso X hasta que ocurre C”, o “Si ocurre C se pasa al paso X” . En estas situaciones lo importante no es la forma en la que se expresan las condiciones e iteraciones, sino hacerlo de una forma consistente. Si la descripción del caso fuera muy compleja, es conveniente usar notaciones gráficas, por ejemplo los diagramas de actividad.

2.3. Alternativas

Durante la ejecución de un caso de uso, suelen aparecer errores o excepciones. Por ejemplo, mientras se ingresa un pedido, el cliente puede solicitar un producto que está discontinuado. El sistema deberá en este caso informar esta situación al empleado que ingresa el pedido. Esas desviaciones del curso normal del caso de uso se llaman alternativas. Las alternativas tienen las siguientes características:

1) Representan un error o excepción en el curso normal del caso de uso.

2) No tienen sentido por sí mismas, fuera del contexto del caso de uso en el que ocurren.

Si bien en la bibliografía las alternativas se documentan al final del caso de uso, la experiencia demuestra que resulta útil documentar los casos en tablas, mostrando el curso principal en la primera columna, y las alternativas en una segunda columna, como lo muestra el siguiente ejemplo:

Caso de Uso: Ingresando Pedido	
Actor: Empleado de ventas	
Curso Normal	Alternativas
1) El cliente se comunica con la oficina de ventas, e informa su número de cliente	
2) El oficial de ventas ingresa el número de cliente en el sistema	
3) El sistema obtiene la información básica sobre el cliente	3.1 Si no está registrado, se le informa que debe registrarse en la oficina de clientes
4) El cliente informa el producto que quiere comprar, indicando la cantidad	
5) El sistema obtiene la información sobre el producto solicitado, y confirma su disponibilidad.	5.1 Si no hay disponibilidad del producto, el sistema informa la fecha de reposición
6) Se repite el paso 4) hasta que el cliente no informa más productos	
...	

Figura 5 – Algunas alternativas del caso de uso “Ingresando Pedido”

De esta forma, es mucho más simple ver en qué parte del caso de uso puede ocurrir la excepción, y se mantiene la ventaja de poder leer de corrido el curso normal.

3. Modularización de Casos de Uso

Las próximas secciones muestran cómo se puede organizar una especificación que utiliza casos de uso para evitar redundancia (duplicación innecesaria de información) y facilitar su comprensión.

3.1. Relaciones de Extensión

Muchas veces, la funcionalidad de un caso de uso incluye un conjunto de pasos que ocurren sólo en algunas oportunidades. Supongamos que estamos especificando un sistema en el cual los clientes pueden ingresar pedidos interactivamente, y que dentro de la funcionalidad del ingreso de pedidos el usuario puede solicitar al sistema que le haga una presentación sobre los nuevos productos disponibles, sus características y sus precios. En este caso, tengo una excepción dentro del caso de uso *Ingresando Pedido*. La excepción consiste en interrumpir el caso de uso y pasar a ejecutar el caso de uso *Revisando Presentación de Nuevos Productos*. En este caso decimos que el caso de uso *Revisando Presentación de Nuevos Productos* **extiende** el caso de uso *Ingresando pedido* y se representa por una línea de trazos desde el caso que 'extiende a' al caso que es 'extendido'.

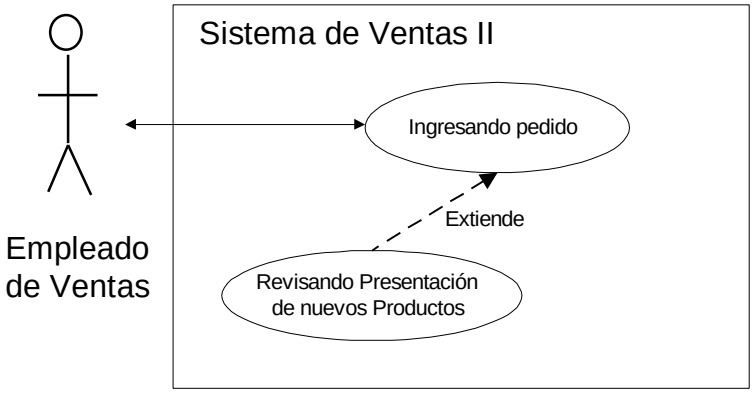

Figura 6 – Una relación de extensión entre dos casos de uso

Las extensiones tienen las siguientes características:

1) Representan una parte de la funcionalidad del caso que no siempre ocurre.
2) Son un caso de uso en sí mismas.
3) No necesariamente provienen de un error o excepción. En su libro, Jacobson ejemplifica los casos de uso con ir a cenar a un restaurant. Para él, tomar café después de cenar es un ejemplo de una extensión.

La pregunta que surge claramente es ¿cuál es la diferencia entre una alternativa y una extensión? La respuesta puede derivarse de las características de cada uno:

- Una extensión es un caso de uso en sí mismo, mientras que una alternativa no.
- Una alternativa es un error o excepción, mientras que una extensión puede no serlo.

De todas formas, en la práctica aparecen dudas con respecto a la conveniencia de considerar algo optativo en un caso como una alternativa o una extensión, sobre todo porque no queda claro si algo puede ser visto como un caso de uso en sí mismo o no. Como regla aproximada en este caso podemos pensar que si algo opcional debe ser expresado con más de un paso, seguramente es una extensión y no una alternativa.

3.2. Relaciones de Uso

Es común que la misma funcionalidad del sistema sea accedida a partir de varios casos de uso. Por ejemplo, la funcionalidad de buscar un producto puede ser accedida desde el ingreso de pedidos, desde las consultas de productos, o desde los reportes de ventas por producto. ¿Cómo hago para no repetir el texto de esta funcionalidad en todos los casos de uso que la acceden? La respuesta es simple: sacando esta funcionalidad a un nuevo caso de uso, que es usado por los casos de los cuales fue sacada. Este tipo de relaciones se llama *relaciones de uso* y se representa por una línea punteada desde el caso que 'usa a' al caso que es 'usado'. Decimos, por ejemplo, que el caso de uso O*bteniendo reporte de ventas por producto* usa al caso de uso *Buscando producto*.

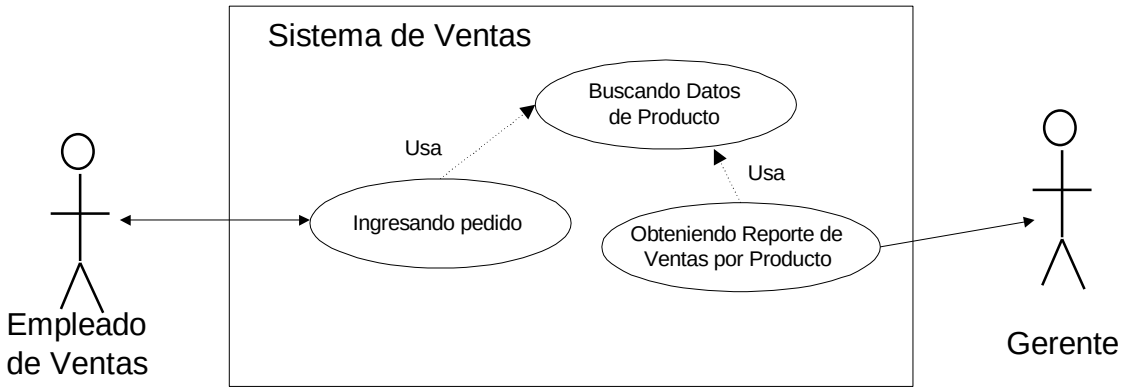

Figura 7 – Relaciones de Uso entre Casos de Uso

Este concepto no es novedoso, es simplemente el concepto de la subrutina o subprograma usado en un nivel más alto de abstracción.

Las características de las relaciones de uso son:

1) Aparecen como funcionalidad común, luego de haber especificado varios casos de uso.
2) Los casos usados son casos de uso en sí mismos.
3) El caso es usado *siempre* que el caso que lo usa es ejecutado. Esto marca la diferencia con las extensiones, que son opcionales.

La definición de las relaciones de uso y extensión deja una zona sin definir:

¿Qué pasa con la funcionalidad que es común a varios casos de uso, pero al mismo tiempo es opcional? Por ejemplo, pensemos en la impresión de un comprobante, algo que el usuario de un sistema puede o no hacer en distintos casos de uso. Si uno se guía por la funcionalidad común a varios casos, piensa que el caso de uso *imprimiendo comprobante* es **usado** por otros casos, pero si se guía por la opcionalidad, piensa que **extiende** a otros casos. Como esto no queda claro a partir de la bibliografía, *creemos conveniente que este tipo de situaciones se especifiquen como extensiones*, ya que de esta forma podemos remarcar gráficamente la opcionalidad de la relación.

3.3. Actores y Casos de Uso Abstractos

Al modularizar la especificación, identificando relaciones de uso y extensión, puede pasar que extraigamos casos de uso que son accedidos por varios actores. Por ejemplo, el caso de uso *buscando datos de producto* es accedido por muchos actores (el empleado de ventas que ingresa un pedido, el gerente que quiere obtener estadísticas por producto, el supervisor que quiere consultar la información de algún producto, etc.). Ahora bien, como el caso de uso nunca se ejecuta fuera del contexto de otro caso de uso, decimos que es un caso de uso *abstracto*. Lo llamamos abstracto porque no es implementable por sí mismo: sólo tiene sentido como parte de otros casos.

De la misma forma, el actor que participa de este caso de uso, que reúne características comunes a todos los actores de los casos de uso que lo usan, es un actor abstracto. En nuestro ejemplo, si bien el nombre suena poco elegante, podemos decir que tenemos un actor abstracto "Buscador de Datos de Producto". Los actores abstractos, entonces, son necesarios para no "dejar sin actores" a los casos de uso abstractos.

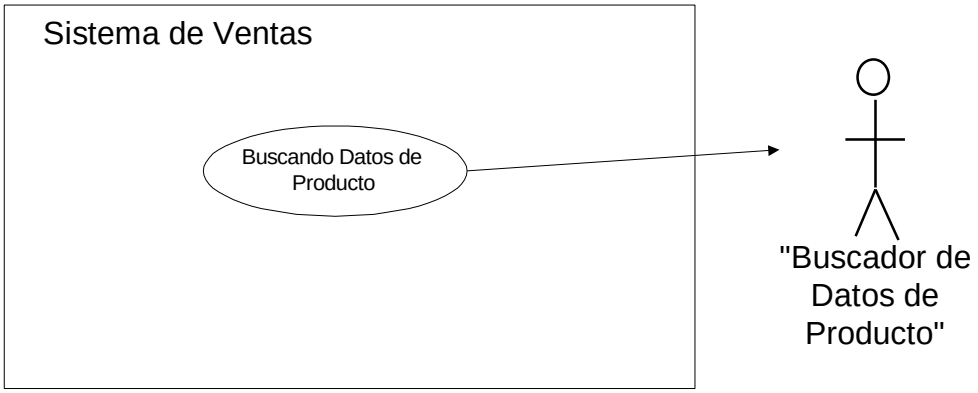

Figura 8 – El nombre de los actores abstractos suele no ser feliz

La duda ahora es cómo relacionar este actor abstracto con los actores concretos: los que sí existen en la realidad y ejecutan casos de uso concretos, como *ingresando pedido* y *obteniendo estadísticas de ventas*.

Para esto podemos usar el concepto de herencia, uno de los conceptos básicos de la orientación a objetos. Como todos los actores concretos también ejecutan el caso *buscando datos de producto*, a través de la relación de uso, podemos decir que los actores concretos *heredan* al actor abstracto.

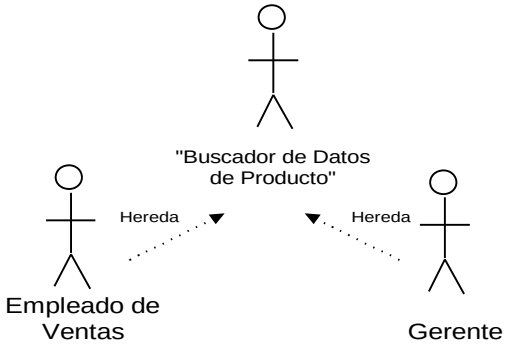

Figura 9 – Herencia de Actores

La relación de herencia no necesariamente implica la existencia de un caso abstracto. Puede ocurrir que un actor ejecute todos los casos que ejecuta otro actor, y algunos más. En nuestro sistema, el supervisor de ventas puede hacer todo lo que hace el empleado de ventas, pero además puede autorizar pedidos. En este caso, podemos decir que el Supervisor de Ventas *hereda* al Empleado de Ventas, aunque el Empleado de Ventas no sea un actor abstracto. De esta forma, toda la funcionalidad que está habilitada para el Empleado de Ventas también lo está para el Supervisor.

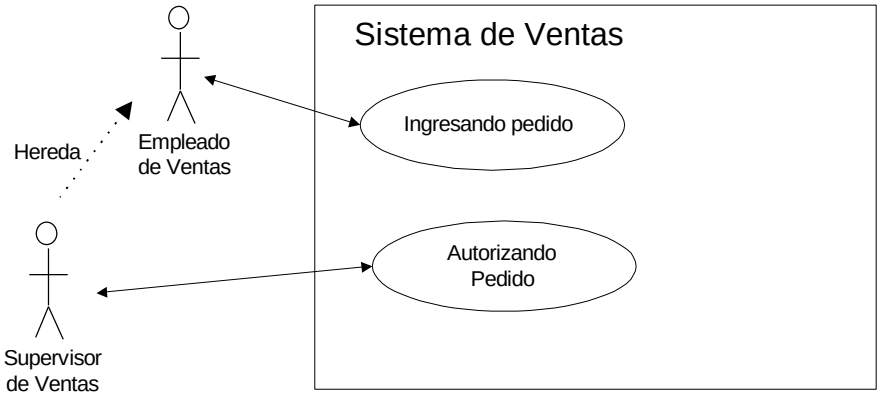

Figura 10 – Relación de herencia entre actores concretos

Usando la herencia en el análisis de requerimientos evitamos redundancia y simplificamos la especificación y los gráficos de casos de uso.

4. Tipos de Casos de Uso

A partir de la aplicación de los casos de uso en la práctica, aparecen distintas clasificaciones, que son útiles según el tipo de sistema o el modelo de ciclo de vida utilizado.

4.1. Esenciales o de Trazo Grueso vs. de Implementación o de Trazo Fino

Uno de los modelos de ciclo de vida de desarrollo de sistemas que más popularidad ha ganado en los últimos años es el llamado "modelo incremental", en el cual se van entregando versiones parciales del sistema, que implementan una parte de su funcionalidad. La recomendación en este caso pasa siempre por identificar todos los requerimientos que uno pueda, definir sus prioridades, y seleccionar cuáles se van a ir implementado en cada versión. Sin embargo, no se pueden especificar en detalle todos los requerimientos: debemos tener apenas el nivel de detalle suficiente para poder definir sus prioridades y comprenderlos en términos generales.

Para aplicar los casos de uso a desarrollos incrementales, empezamos por identificar todos los casos de uso del sistema, sólo al nivel de su nombre. Una vez que los identificamos, los expresamos en "trazo grueso", esto es:

- Ignoramos detalles sobre la forma de la interacción entre el actor y el sistema.
- Sólo incluimos las alternativas más relevantes, ignorando la mayoría de los errores que aparecen en el uso del sistema.
- No entramos en detalle sobre las acciones que realiza el sistema cuando el usuario interactúa con él. Por ejemplo, si la empresa tuviera una política de descuentos para sus clientes, no es necesario especificar cómo es esa política: nos alcanza con saber que existe una y que debe ser tenida en cuenta.

De esta forma, terminamos con una descripción "gruesa" de todos los casos de uso. Esto me sirve para tomar mejores decisiones, junto con los usuarios, sobre qué casos de uso implementar en cada fase. Por otro lado, permite analizar los principales aspectos de todos los casos que afectan al diseño.

Los casos de uso de trazo fino son aquellos que se especifican una vez que se ha tomado la decisión de implementarlos. En este momento debemos completar todos los detalles que dejamos pasar:

- A medida que vamos haciendo prototipos de las interfaces con los usuarios, incluimos detalles sobre la forma de la interfaz en la descripción del caso. Por ejemplo, podemos incluir detalles como: "el operador puede en cualquier momento pasar de la ventana de datos del cliente a la ventana de datos del pedido". Si bien esto implica anticiparse al diseño, esto no es negativo, ya que es prácticamente imposible –y perjudicial– hablar con los usuarios siempre en términos de un sistema abstracto.
- Incluimos otras alternativas. En particular especificamos todos los errores o excepciones que provienen de requerimientos de los usuarios. Para esto debemos tener en cuenta que un sistema tiene dos tipos de errores o excepciones: las que provienen de las definiciones del negocio y las que provienen del procesamiento interno del sistema. Por ejemplo, pensemos en un requerimiento del tipo: "si un cliente hace un pedido por un monto mayor al autorizado, se debe rechazar el pedido". Esta excepción es claramente un requerimiento, y debe ser incluida en las alternativas de los casos de uso. Por el contrario, una excepción del tipo: "Si el usuario ingresa una letra en el lugar del código del producto se le informa que el código de producto debe ser numérico" no debe ser incluida en esta etapa del análisis.
- Especificamos con más detalle el comportamiento interno del sistema. En nuestro ejemplo de los descuentos, deberíamos especificar cómo es esa política, en un nivel de detalle suficiente para luego poder diseñar una forma de implementarla dentro del sistema.

4.2. Casos de Uso Temporales

Los casos de uso tienen un actor que los inicia, y uno o más actores que participan de él. En muchos casos, el inicio de una determinada funcionalidad del sistema es provocado exclusivamente por el paso del tiempo. Supongamos que nuestro sistema de ventas debe generar en forma automática un conjunto de estadísticas para ser entregadas al directorio de la empresa el último día hábil de cada mes. En este caso, el paso del tiempo es el que inicia el caso de uso, y el directorio es el actor del sistema. Sin embargo, para expresar claramente que es el paso del tiempo el que inicia el caso, podemos incluir un símbolo representando un reloj en el gráfico de casos de uso, o usar una línea punteada en el borde del óvalo del caso.

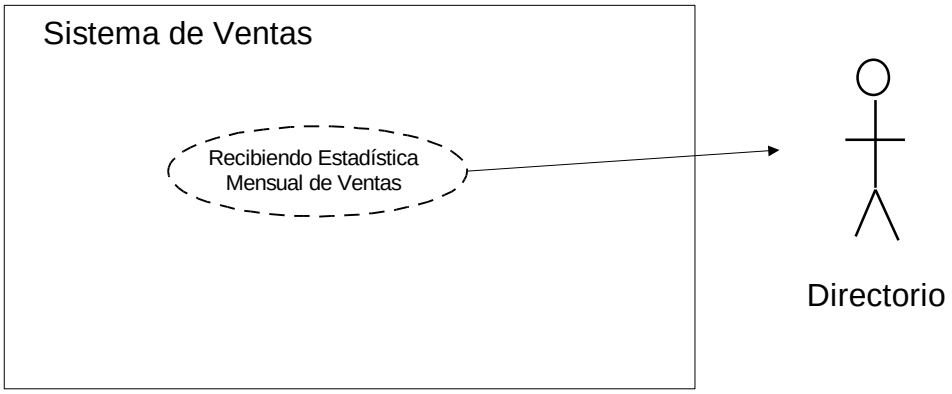

Figura 11 – Un caso de uso temporal

Es importante que cuando se especifican los casos de uso de trazo fino, se exprese claramente cuál es el momento del tiempo en el que se inicia el caso. Es común que se indiquen cosas como "una vez al mes" cuando se habla de casos iniciados por el paso del tiempo. Cuando hacemos los casos de trazo fino, debemos precisar en qué momento del mes eso ocurre (el primer día hábil, el primer día calendario, el último día hábil, etc.). De lo contrario, estamos dejando en los diseñadores la decisión sobre cuándo generar esta salida, y esto no es correcto, ya que la oportunidad de las salidas del sistema debe ser definida por los usuarios.

4.3. Casos Primarios Vs. Casos Secundarios

Jacobson hace referencia a la diferencia entre los casos de uso primarios del sistema y aquellos que no se corresponden con procesos del negocio y cuya ejecución sólo es necesaria para que el sistema funcione normalmente.

Supongamos que nuestro sistema requiere de un proceso de depuración de los pedidos que ya han sido cumplidos hace más de 5 años, para evitar que se acumulen indefinidamente. El caso de uso por el cual se depuran estos pedidos, cuyo actor es un Administrador del Sistema, es considerado un caso secundario, ya que no es central al sistema, sino que es necesario para que el sistema pueda funcionar sin problemas. En la experiencia se ve que no todos los casos de uso secundarios se pueden identificar en la etapa de requerimientos, ya que muchos de ellos dependen de decisiones de implementación que se toman en la etapa de diseño. De todas formas, los casos de uso pueden ser actualizados a medida que progresa el desarrollo del sistema, ya que al estar expresados desde el punto de vista del usuario, son una excelente base para construir del manual de usuario.

5. El Proceso de Análisis de Requerimientos con Casos de Uso

Esta sección describe los pasos a seguir para aplicar la técnica de análisis de requerimientos con casos de uso.

5.1. Identificar los Actores

Si la primera pregunta que un analista debe hacer a sus usuarios es ¿Para qué es este sistema?, la segunda es claramente ¿Para quiénes es este sistema? Como mencionamos al hablar sobre los actores, identificar a todos ellos es crítico para un buen análisis de requerimientos. Por lo tanto, antes de avanzar con los casos de uso, debo tratar de identificar todos los tipos de usuario diferentes que tiene el sistema. Si el sistema funcionará en una empresa, debo preguntar cuáles de las áreas afectadas usarán o actualizarán su información.

A pesar de hacer una identificación inicial de los actores, también debo repetirla a medida que empiezo a describir los casos de uso, ya que al conocer más detalles del sistema pueden aparecer nuevos tipos de usuarios.

5.2. Identificar los Principales Casos de uso de Cada Actor

El siguiente paso es enunciar los nombres de los principales casos de uso de cada uno de los actores que identifiqué en el paso anterior. No es necesario especificar cuáles son las acciones dentro del caso de uso. Tampoco debo preocuparme si no aparecen muchos casos, ya que existen técnicas para encontrar nuevos casos de uso a partir de los existentes.

5.3. Identificar Nuevos Casos a Partir de los Existentes

Uno de los principales errores que se pueden cometer al identificar requerimientos es algo que parece obvio, pero que muchas veces ocurre: ¡olvidarse de algún requerimiento! Como los requerimientos están en la cabeza de los usuarios, el éxito de esta tarea depende de la habilidad del analista. Para ayudarnos a identificar nuevos casos de uso a partir de los casos existentes, podemos aplicar las mismas técnicas utilizadas para identificar eventos según el análisis estructurado. Esta técnica se basa en el análisis de cuatro situaciones posibles a partir de los requerimientos ya identificados.

Variaciones Significativas de Casos de Uso Existentes

Muchas veces existen variaciones en los casos de uso en función del actor que los ejecuta, o del tipo de objeto sobre el que se apliquen. Por ejemplo, en el caso del sistema que procesa pedidos, podemos hacernos las siguientes preguntas:

1) ¿Existen distintos tipos de cliente que hagan pedidos?
2) ¿Existen distintos tipos de pedidos, que lleven a acciones distintas por parte del sistema?

Asumiendo que la respuesta a la primera pregunta es si, lo próximo que debemos preguntarnos es si el sistema ejecuta acciones distintas en función del tipo de cliente. Tal vez la respuesta sea que existen clientes locales y clientes del exterior, y que en estos dos casos el procesamiento es totalmente diferente, ya que los pedidos del exterior deben ser procesados por le Gerencia de Comercio Exterior. Tal vez estemos frente a un nuevo caso de uso, "Ingresando Pedido de Cliente del Exterior", que es distinto del caso de uso "Ingresando Pedido de Cliente Local". Tal vez tengamos dudas sobre si estos son dos casos de uso o uno solo, porque el proceso puede tener puntos en común y puntos que los distinguen. En esta última situación me veo obligado a usar nuevamente el sentido común. Tal vez aplicando la modularización de casos de uso a través de las relaciones de uso, puedo factorizar la funcionalidad común y expresar claramente, incluso gráficamente, la funcionalidad que los distingue.

Por supuesto que si la respuesta a la primera pregunta es no, estamos en el caso en que no vamos a encontrar un nuevo caso de uso a partir de este análisis.

Supongamos ahora que la respuesta a la segunda pregunta es sí. En este caso, nuevamente podemos encontrar un nuevo caso de uso. Por ejemplo, podemos encontrar que hay muchas diferencias entre el procesamiento de pedidos de ciertos tipos de productos. En este caso, nuevamente debemos decidir si la funcionalidad diferenciada es lo suficientemente relevante como para especificar un nuevo caso. Para hacer este análisis, debemos tener en cuenta lo siguiente:

1) Si especificamos dos casos de uso similares como un único caso de uso, en el texto del caso tendremos muchos "Si pasa X, hago A, si no, hago B". Este hace un poco más difícil de seguir la especificación.

2) Si especifico dos casos de uso con funcionalidad en común como dos casos de uso distintos, la relación de uso me puede ayudar a evitar la redundancia.

De todas formas, no debo llevar estas reglas al extremo, como por ejemplo buscar que todos los casos sean lineales (sin decisiones), ya que de esta forma lo único que voy a conseguir es una maraña incomprensible de casos de uso.

Casos de Uso "Opuestos"

Hay dos formas de buscar casos de uso opuestos. La primera es buscar la función opuesta a la descrita por el caso. Por ejemplo, en el caso de realizar un pedido, podemos pensar que la función opuesta es cancelar ese mismo pedido. Si cancelar pedidos es una función que mi sistema debe realizar, y que no había identificado anteriormente, acabo de evitarme un error que puede ser muy costoso de corregir más adelante.

La otra forma de buscar casos de uso opuestos es pensar en la negación de la acción principal del caso de uso. Supongamos que tenemos un caso de uso "Pagando Pedido", que es realizado por el actor cliente. El caso "No Pagando Pedido", puede de nuevo ser significativo. Si el cliente no paga el pedido, tal vez nuestro sistema deba hacer algo, y podemos estar frente a un nuevo caso de uso.

Casos de Uso que Preceden a Casos Existentes

Una buena pregunta para hacer frente a un caso de uso es:

¿Qué es lo que tiene que ocurrir antes de este caso de uso?

En el caso del cliente que hace un pedido, son muchas las cosas que pueden ocurrir antes de ese caso:

1) El cliente, por ejemplo, debe ser cliente. En esta situación tal vez tenga un nuevo caso de uso "Ingresando Cliente", que puede o no ser un caso de mi sistema.

2) El cliente debe poder consultar cuáles son los productos existentes. Probablemente este sea un caso de uso que ya haya sido identificado. Sin embargo, usando esta técnica muchas veces se descubren nuevos requerimientos.

Casos de Uso que Suceden a Casos Existentes

Esto es similar al punto anterior. La pregunta que debo hacerme es:

¿Qué ocurre después de este caso de uso?

En nuestro ejemplo de los pedidos, es evidente que la mayoría de la funcionalidad de nuestro sistema recién empieza cuando el cliente hace un pedido. Por lo tanto, analizar "cómo sigue la historia" es una buena forma de asegurarme que no estoy dejando requerimientos sin identificar.

5.4. Crear Descripciones de Casos de Uso de Trazo Grueso

Una vez que identificamos todos los casos de uso, empezamos a documentar sus pasos. Esta tarea no es estrictamente secuencial de la anterior: es posible que, mientras empezamos a documentar los casos, sigamos buscando otros nuevos.

La documentación de los casos de uso identificados debe hacerse del tipo "trazo grueso", salvo que sea un caso de uso que deba implementarse *sí o sí* en la primera iteración.

5.5. Definir Prioridades y Seleccionar Casos de la Primera Iteración

Una vez documentados los casos de trazo grueso, es conveniente definir las prioridades de los distintos requerimientos, expresados como casos de uso. Para esto suele ser útil usar tres categorías: *imprescindible, importante y deseable.*

- Los requerimientos imprescindibles son aquellos que, si no se implementan, hacen que el sistema no tenga sentido.
- Los importantes son aquellos que harían que el usuario se sienta decepcionado si no se implementan.
- Los deseables son aquellos que el usuario querría tener, si hubiese tiempo disponible.

Al evaluar un requerimiento debo también analizar su costo o complejidad.

Una vez hecha esta categorización de los requerimientos, puedo tomar como estrategia general el incluir los imprescindibles, discutir los importantes y descartar los deseables cuyo costo no sea bajo. Por lo dicho anteriormente, esta regla también cumple con la regla de ser relativa pues debo analizar su costo, complejidad, y una cantidad de otros factores antes de decidir su inclusión. Por ejemplo, si un requerimiento fuera trivial de implementar, puede ser una buena idea incluirlo por más que éste sea sólo deseable.

5.6. Escribir los Casos de Trazo Fino y Crear Prototipos de Interfaces

Una vez seleccionados los casos de uso que pienso implementar en la primera iteración, empiezo a profundizar sus definiciones. Al mismo tiempo, suele ser una excelente idea crear un prototipo visual de la implementación de los casos. Con las herramientas existentes actualmente, esto es muy simple de hacer, y nos puede evitar muchos problemas.

Cuando creamos prototipos de la implementación de los casos de uso, debemos tener en cuenta que:

1) **Estos son prototipos para descartar**: si incluimos algo de código en el prototipo, debe ser descartado o revisado con mucho cuidado si se lo piensa mantener.
2) **Estamos intentando validar el estilo de la interacción, no toda la interacción**: no debemos dejarnos llevar por el entusiasmo. Si tenemos varios casos de uso que realizan funciones similares, o con una interfaz similar, puede alcanzar con crear un prototipo de sólo una de ellas. Queremos que el usuario se dé una idea sobre cómo se verá el sistema, no que nos apruebe todas sus pantallas y listados.

6. Organización de la Especificación

En esta sección discutimos la mejor forma de organizar una especificación de requerimientos en la que se aplicó la técnica de casos de uso.

6.1. Gráficos a Utilizar

Dependiendo del tamaño del sistema, es probable que un único gráfico con todos los casos de uso nos quede chico. No olvidemos que los modelos gráficos son para aclarar el texto, y no para confundir. Si el gráfico de casos de uso es una maraña indescifrable, no está cumpliendo su objetivo. Por lo tanto, podemos usar las siguientes reglas, como siempre con criterio y sentido común:

1) Un gráfico de casos de uso no debe mostrar más de 15 casos
2) Si debo particionar mi gráfico, puedo hacerlo por actor. La primera partición debe ser separar los casos centrales de los casos auxiliares, ya que probablemente les interesen a personas distintas.
3) Si las relaciones de uso y las extensiones entran en el diagrama principal, sin dejar de cumplir con la regla 1), debo dejarlas ahí. Lo mismo se aplica a los actores abstractos.
4) Si las relaciones de uso no entran en el diagrama principal, debo mostrarlas en gráficos teniendo en cuenta que siempre debo mostrar todos los casos de uso que usan a otro en un mismo diagrama.
5) Si tengo un caso de uso que es usado por gran parte de los otros casos, como por ejemplo el caso de uso *Identificándose ante el sistema*, debo evitar mostrarlo en el gráfico principal, ya que las flechas serán imposibles de organizar. Es probable que no haga falta mostrar esta relación de uso en un gráfico.

6.2. Secciones de la especificación

Sugerimos el siguiente orden para una especificación de requerimientos utilizando casos de uso:

1) Propósito del sistema: un breve párrafo, de 4 o 5 líneas, que responde a la pregunta ¿Para qué estamos haciendo este sistema?
2) Gráfico(s) de casos de uso
3) Descripción de los casos con sus alternativas
4) Prototipos para los principales casos de uso

Esta no es obviamente una especificación de requerimientos completa: estamos incluyendo sólo la parte referida a los casos de uso.

7. Bibliografía

[Brooks87] Frederik P. Brooks - *No Silver Bullet. Essence and Accidents in Software Engineering*. IEEE Computer. Abril 1987.

[Jacobson92] Ivar Jacobson y otros. *Object Oriented Software Engineering. A Use Case Driven Approach*. Addison Wesley, 1992.

[McMenamin84] Steve Mc. Menamin y John Palmer. *Essential Systems Analysis*. Prentice Hall, Yourdon Press, 1984.