La Especificación de Requisitos con Casos de Uso: Buenas y Malas Prácticas

José Antonio Pow Sang Portillo
Pontificia Universidad Católica del Perú
Av. Universitaria cdra. 18, San Miguel
Lima-32- PERU
japowsang@pucp.edu.pe

Resumen. El uso de UML como estándar para la construcción de software se ha extendido en los últimos años. Es por eso que el empleo de los casos de uso, como parte del estándar UML, se ha incrementado.
El propósito de los casos de uso es describir en lenguaje natural la funcionalidad completa de un sistema a desarrollar y su empleo se realiza en el proceso de especificación de requisitos del sistema.
Lamentablemente, la bibliografía existente, muestra muchas formas de aplicar los casos de uso y no son pocas las veces en que su empleo es inadecuado. Algunas de las causas son: mala interpretación del estándar UML y secuencia incorrecta de actividades para la creación de casos de uso.
Este artículo presenta un esquema de trabajo para afrontar el proceso en mención utilizando casos de uso. Se incluye, en este esquema, las actividades que se deben realizar, la utilización correcta de casos de uso y los errores que se cometen frecuentemente en cada una de las actividades.
Cabe resaltar que este esquema de trabajo es aplicado en los proyectos que forman parte de los cursos del área de Ing. de Software de la PUCP a nivel pre y post grado. También, es utilizado en las tesis para optar el título de Ing. Informático.

Palabras claves: Especificación de Requisitos de Software, Ingeniería de Requisitos, Casos de Uso, Escenarios.

Abstract. The use of UML as a standard to construct software has been increased over the last years. For this reason, the utilization of use cases has been growing.
The purpose of the use cases is to describe software functionality using natural language. Use cases are used in software requirement specification process.
Unfortunately, the authors show many ways to apply use cases and their use is not according to UML. Sometimes people misunderstand UML standard and follow an inadequate sequence of activities to obtain requirements with use cases.
This article shows a method to face requirements process with use cases. This article also includes the correct use of use cases and common mistakes. The method is used in undergraduate and post-graduate Software Engineering courses at PUCP.

Keywords: Software Requirements Specification, Requirements Engineering, Use Cases, Sceneries.

1. Introducción

Uno de los primeros procesos que se realizan en un proyecto de construcción de software es la especificación de requisitos. Los objetivos de este proceso son identificar, validar y documentar los requisitos de software; es decir determinar las características que deberán tener el sistema o las restricciones que deberá cumplir para que sea aceptado por el cliente y los futuros usuarios del sistema de software.

El producto final de este proceso es el documento de especificación de requisitos de software y en éste se señala, con el detalle adecuado, lo que el usuario necesita del sistema de software. Es por ello, que el documento de requisitos de software se considera como un contrato entre el cliente y el equipo de desarrollo del sistema.

Actualmente, el desarrollo de software orientado a objetos y el uso de UML se han incrementado. Es por ese motivo que el empleo de casos de uso se está imponiendo frente a otras técnicas de especificación de requisitos.

Lamentablemente, la bibliografía existente muestra muchas formas de aplicar los casos de uso y no son pocas las veces en que su empleo es incorrecto. Algunas de las causas son: mala interpretación del estándar UML y secuencia incorrecta de actividades para la creación de casos de uso.

Este artículo muestra las actividades y la secuencia a seguir para realizar una especificación de requisitos empleando casos de uso. Además, se explicarán los errores comunes que se producen en cada una de esas actividades.

2. UML y la Especificación de Requisitos

Para determinar la funcionalidad de un sistema a desarrollar, UML [8] señala el uso de dos elementos: el actor y el caso de uso.

El actor representa una entidad externa que interactúa con el sistema. Las entidades externas podrían ser personas u otros sistemas. Es importante resaltar que los actores son abstracciones de papeles o roles y no necesariamente tienen una correspondencia directa con personas.

A diferencia del actor, el caso de uso hace referencia al sistema a construir, detallando su comportamiento, el cual se traduce en resultados que pueden ser observados por el actor. Los casos de uso describen las cosas que los actores quieren que el sistema haga, por lo que un caso de uso debería ser una tarea completa desde la perspectiva del actor.

Los actores y los casos de uso forman un modelo al que se le denomina "modelo de casos de uso". Dicho modelo muestra el comportamiento del sistema desde la perspectiva del usuario y servirá como producto de entrada para el análisis y diseño del sistema. La figura 1 muestra la notación que se debe utilizar para representar un actor y un caso de uso.

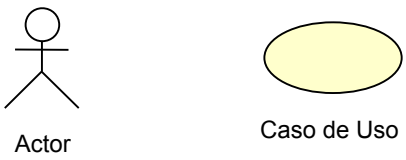

Figura 1. Representación gráfica de actor y caso de uso

UML especifica que para representar gráficamente la relación entre un actor y caso de uso se debe trazar una línea que los una a la que se le denomina "relación de comunicación". Además, UML señala que los casos de uso pueden tener relaciones entre sí. Los tipos de relaciones que pueden existir son: "include", "extends" y "generalization". La figura 2 muestra un ejemplo de casos de uso con relaciones de tipo "generalization".

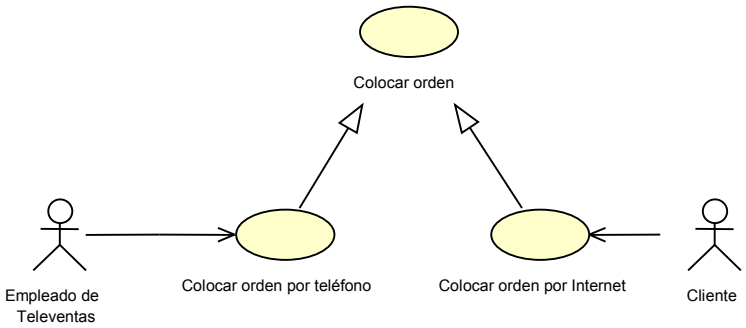

Figura 2. Diagrama de casos de uso con relación "generalization" entre casos de uso.

3. Actividades para la Especificación de Requisitos con Casos de Uso

Los resultados de la especificación de requisitos son dos productos: el catálogo de requisitos y el documento de especificación de requisitos de software. El primero de ellos contiene la lista de requisitos de software clasificada por tipo y prioridad; y el segundo de ellos, especifica el comportamiento del sistema a un grado de detalle mayor al del catálogo de requisitos. La figura 3, indica el contenido de los productos de la especificación de requisitos de software mediante un diagrama de clases.

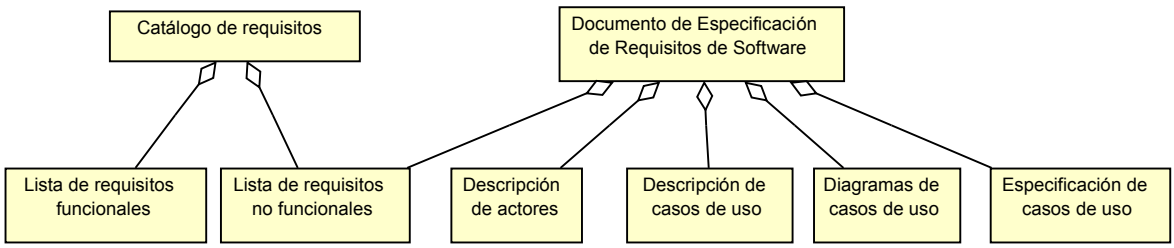

Figura 3. Diagrama de clases que representan los productos de la especificación de requisitos

Las actividades para la especificación de requisitos de software usando casos de uso son las siguientes: identificar y clasificar requisitos, identificar actores, identificar escenarios, identificar casos de uso, especificar casos de uso e identificar relaciones entre casos de uso. La secuencia de actividades, que se detallan en este acápite, es el resultado de la adaptación de las propuestas metodológicas de Bernd Bruegge [3], Rational Unified Process [9] y Métrica Versión 3 [7] para la especificación de requisitos. La figura 4 muestra la secuencia en que se deben realizar las actividades y sus resultados.

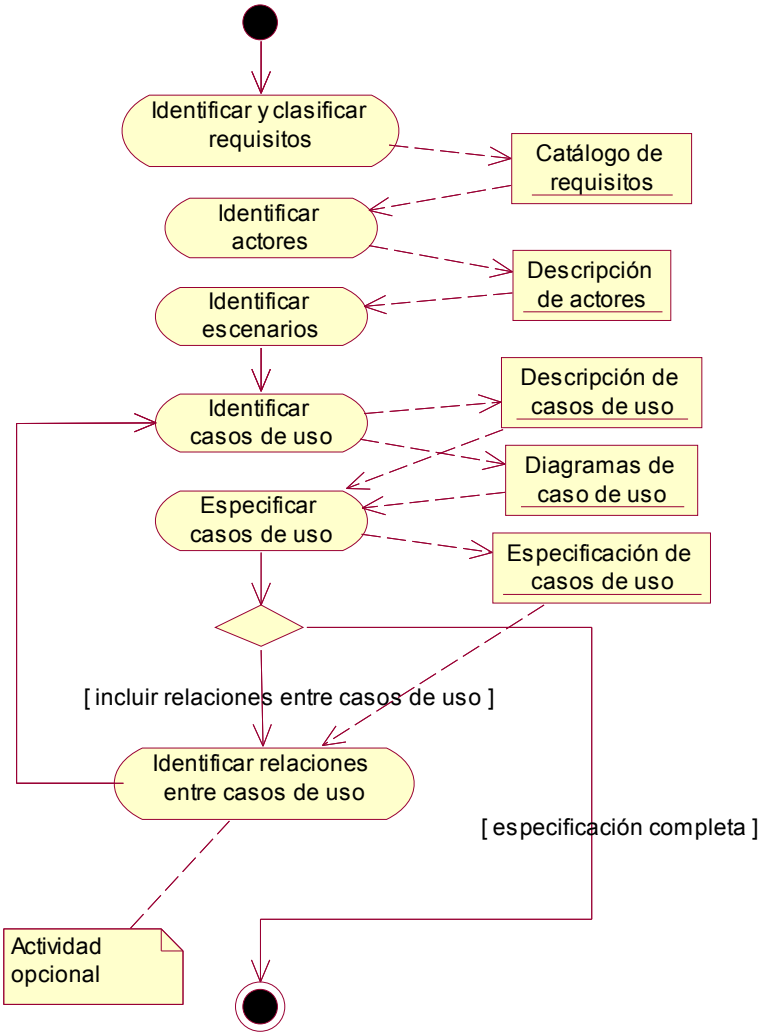

Figura 4: Actividades para la especificación de requisitos usando casos de uso.

A continuación se explica cada una de las actividades propuestas.

Actividad 1: Identificar y clasificar requisitos.

Esta actividad es el punto de partida para las siguientes actividades del proceso de obtención de requisitos y se refiere a la identificación de los requisitos del sistema de software a desarrollar.

En esta actividad, deberemos responder a los siguientes cuestionamientos: ¿qué le permitirá hacer, el sistema de software, al usuario? y ¿el cliente o usuario me solicita alguna restricción para construir el sistema de software? Contestando a esas preguntas se deberá realizar una lista que contendrá los requisitos del sistema.

Luego de haber obtenido la lista de requisitos, estos deberán ser clasificados en dos grupos: requisitos funcionales y requisitos no funcionales. Los requisitos funcionales son declaraciones de los servicios que proveerá el sistema, de la manera en que éste reaccionará a entradas particulares y de cómo se comportará en situaciones particulares. En algunos casos, los requisitos funcionales declaran explícitamente lo que el sistema no debe hacer.

A diferencia de los requisitos funcionales, los no funcionales no se refieren directamente a las funciones específicas que entrega el sistema, sino a sus propiedades como fiabilidad, respuesta en el tiempo y capacidad de almacenamiento. Los requisitos funcionales son restricciones de los servicios o funciones ofrecidos por el sistema.

La tabla 1 muestra una relación de requisitos funcionales y no funcionales

Tabla 1. Ejemplo de Requisitos funcionales y no funcionales

Requisitos Funcionales	Requisitos No Funcionales
1. El sistema permitirá registrar los clientes de la empresa. 2. El sistema permitirá a los usuarios realizar una búsqueda de los clientes por DNI, nombre o apellido.	3. La interfaz de usuario del sistema se implementará sobre un navegador Web 4. El sistema deberá soportar al menos 20 transacciones por segundo 5. El sistema permitirá que los nuevos usuarios se familiaricen con su uso en menos de 15 minutos.

Luego, estos requisitos se clasificarán según su importancia, obteniéndose, de esta manera, una lista que contendrá los requisitos clasificados por dos criterios: tipo de requisito (funcional y no funcional) e importancia. A esta lista se le conoce como catálogo de requisitos.

Actividad 2: Identificar actores.

Luego de haber identificado los requisitos funcionales y no funcionales se procederá a identificar los actores del sistema. Para encontrar actores del sistema se puede buscar en las categorías de personas, otro software, dispositivos de hardware o redes de computadoras.

Para un sistema de biblioteca, los actores podrían ser: bibliotecario y cliente (si es que hay módulos de consulta de libros). En el caso de un sistema de ventas, los actores podrían ser: el cliente (si se realiza ventas por Internet), el vendedor y el sistema de facturación.

En un sistema, un usuario del sistema puede actuar como muchos actores; por ejemplo, en un banco, Juan Pérez podría ser cliente y operador dependiendo el momento y el uso que haga del sistema.

Actividad 3: Identificar escenarios.

Un escenario, según Bruegge [3] "es una descripción concreta, enfocada e informal de una sola característica del sistema desde el punto de vista de un solo actor"; es decir, un escenario muestra la secuencia de pasos que se produce cuando un actor interactúa con el sistema en una situación específica y un tiempo determinado.

Un ejemplo de escenario para un sistema de biblioteca es el siguiente: "Juan Pérez se conecta al sistema de la Biblioteca Nacional a través de Internet. Juan Pérez selecciona realizar búsqueda y cuando aparece el formulario ingresa en título de libros la frase 'especificación de requisitos'. El sistema encuentra un único libro y lo muestra, el libro de la biblioteca es 'Especificación de Requisitos de Software' de Alan Davis y código B 73-825"

Cabe resaltar que no es necesario documentar los escenarios de manera formal. Esto quiere decir que carece de importancia la creación de documentos que describan todos los escenarios posibles del sistema, ya que su propósito es servir en la identificación de los casos de uso del sistema.

Actividad 4: Identificar casos de uso.

La diferencia entre escenarios y casos de uso radica en que un escenario es una instancia de un caso de uso [3]. El caso de uso es el que especifica todos los escenarios posibles para una parte de funcionalidad dada; es decir, todos los escenarios similares se agrupan en un solo caso de uso.

Por ejemplo, en el sistema de biblioteca las consultas de bibliografía por Internet por parte de Juan Pérez y cualquier otro cliente se puede agrupar en un solo caso de uso, al que se le puede denominar "Consultar bibliografía" (Ver figura 5).

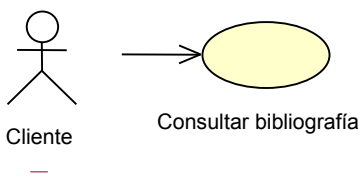

Figura 5. Diagrama de casos de uso para un sistema de biblioteca

Actividad 5: Especificar de casos de uso

Luego de haber identificado los casos de uso, se tienen que indicar, detalladamente, la forma en la que el actor

interactúa con el sistema. Esto se determina mediante la especificación y documentación de cada caso de uso.
RUP [9]precisa que la especificación de cada caso de uso debe contener lo siguiente:

1. Precondiciones, que señalan los estados en que debe estar el sistema para que se pueda ejecutar el caso de uso.
2. Flujo básico, que señala la secuencia de pasos que se va a producir en la mayoría de las veces en que ese caso de uso se ejecute.
3. Flujos alternativos, que contienen las secuencias de pasos que se producirán como alternativas al flujo básico del caso de uso; es decir, especifican los pasos que se producirán en situaciones excepcionales.
4. Postcondiciones, que señalan el estado en que el sistema quedará luego de haberse ejecutado el caso de uso.

Es importante resaltar que durante esta actividad se pueden producir las siguientes situaciones que deben tenerse en cuenta y que no deben ser motivo de preocupación:

1. Un caso de uso que debe partirse en dos casos de uso.
2. Un caso de uso que debe eliminarse, ya que debería formar parte de otro caso de uso.
3. Dos casos de uso que deben formar uno solo.

Actividad 6: Identificar relaciones entre casos de uso (opcional).
En esta actividad se identifican, en base a las especificaciones de casos de uso, las relaciones "include", "extends" y "generalization" entre casos de uso. Es importante resaltar que esta actividad es opcional.

La relación "include" se deberá determinar cuando la especificación de dos o más casos de uso contenga secuencias de acciones iguales. Para ello, la secuencia de pasos que se repite entre ellos, será extraída de esos casos de uso y se creará un nuevo caso de uso que los incluya.

La relación "extend" se produce cuando existe una secuencia de acciones que se producen en ocasiones excepcionales. Para ello, se creará un nuevo caso de uso que contenga dicha secuencia de pasos y dicho caso de uso extenderá la funcionalidad del caso uso original.

En cuanto a la relación "generalization" se identifica cuando existen casos de uso cuyo propósito es similar y contienen secuencias de acciones parecidas. En ese caso, se crea un caso de uso genérico al cual se le denomina caso de uso padre, del cual heredan dos o más casos de uso. La relación "generalization" entre casos de uso es análoga a la relación de "herencia" entre clases.

4. Errores Comunes en la Especificación de Requisitos usando Casos de Uso

En cada una de las actividades especificadas anteriormente se producen y generan errores. A continuación se incluyen algunos de los más frecuentes.

4.1 Errores en la identificación de actores

Los errores introducidos en esta etapa se deben principalmente a no comprender quiénes son los actores del sistema. En algunos casos se incluyen actores que realmente no lo son; por ejemplo, en un sistema en el que se realizan pedidos de productos, se considera al cliente como un actor (Ver figura 6). Realmente quien ingresa los pedidos en el sistema es el vendedor y no el cliente, por lo tanto el vendedor sería el actor del sistema.

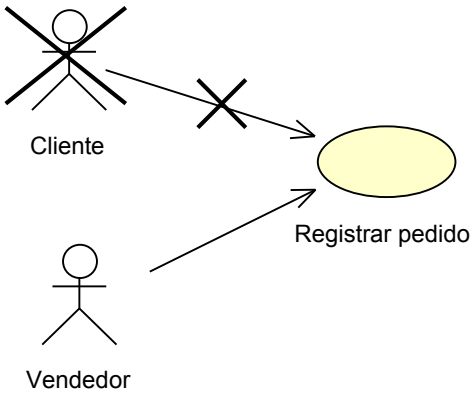

Figura 6. Diagrama de caso de uso para el registro de pedidos.

En el ejemplo de la figura 6, si el sistema permitiera registrar los pedidos por Internet, el cliente sí sería un actor del sistema.

4.2 Errores en la identificación de casos de uso

Un error muy extendido, y que es cometido en la mayoría de la bibliografía sobre casos de uso, es considerar las opciones del menú o funciones del sistema como casos de uso (puede revisar el libro de Larman [6] y podrá encontrar este tipo de errores).

Kurt Bittner [1]señala que los casos de uso deben mostrar lo que el usuario necesita del sistema y no mostrar las funciones u opciones del menú que permitirán realizar lo solicitado; por ejemplo, en un sistema donde se debe almacenar la información de los clientes, lo que al usuario le importa es actualizar la información de clientes. Esta actividad la podrá realizar accediendo a las opciones del menú agregar, modificar y eliminar clientes; por lo tanto la funcionalidad del sistema será representada con el caso de uso "Actualizar cliente" (ver figura 7).

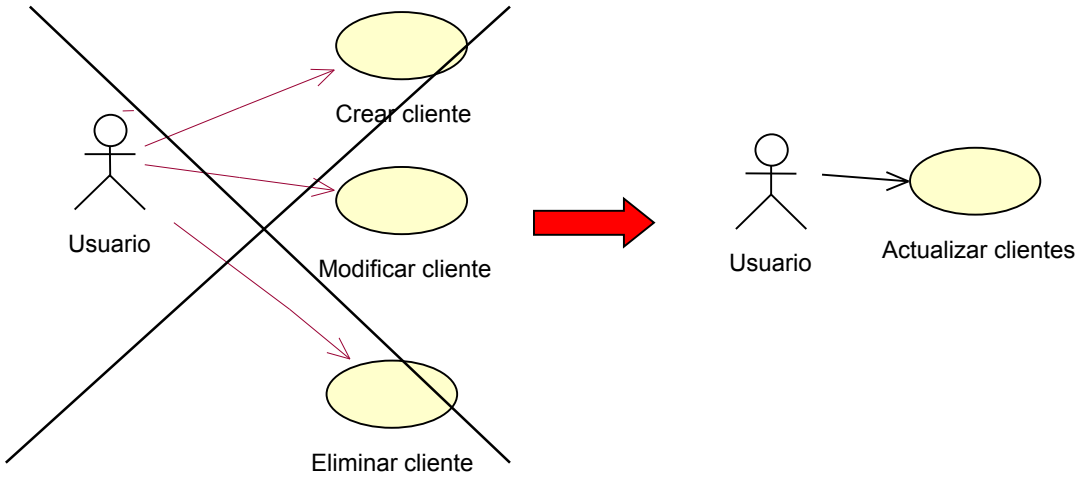

Figura 7. Considerar que los casos de uso son funciones del sistema.

4.3 Errores en la especificación de los casos de uso.

La técnica de casos de uso se debe utilizar para la especificación de requisitos del sistema, mas no para el diseño del sistema. Los errores que se producen en esta actividad se deben a la inclusión de cuestiones de diseño en la especificación de casos de uso. Algunos de los errores que se cometen son los siguientes:

- Introducir palabras que se refieran a componentes de ventanas como: botones, listas desplegables, opciones de menú, etc. En la especificación de casos de uso debe incluirse la información que será ingresada o será mostrada, pero no qué componente de la ventana se va a utilizar para mostrar dicha información, sino se estaría realizando el diseño de pantallas en el proceso de especificación de requisitos, lo cual sería incorrecto.
- Mencionar elementos correspondientes al diseño de algoritmos o de base de datos en la especificación de casos de uso; por ejemplo, "grabar en la tabla clientes en la base de datos" u "ordena los datos con el algoritmo de la burbuja" son oraciones que no deben incluirse en una especificación; ya que son elementos que se determinan en la etapa de diseño.

Otro error es incluir "etc" o "así sucesivamente" cuando se indica la información que se debe ingresar o mostrar. La especificación de casos de uso debe contener información exacta y precisa que permita realizar una buena estimación del esfuerzo requerido para realizar las etapas de análisis, diseño y codificación. Si la información no es exacta, se pueden producir retrasos debido a modificaciones de la base de datos, cambios en el diseño de las pantallas o en el código fuente producto de las especificaciones tardías de requisitos.

4.4 Errores en el uso de las relaciones entre casos de uso.

Los errores que se producen al incluir relaciones entre los casos de uso se deben principalmente a confundir los casos de uso con los procesos de los diagramas de flujo de datos (DFD) de Yourdon [11]. Es por eso que se ven diagramas de casos de uso que parecen DFDs, de manera similar al diagrama que se muestra en la figura 8.

Para evitar cometer este error, se aconseja que no haya más de dos niveles de relaciones de tipo "include" o "extend" en un diagrama de casos de uso.

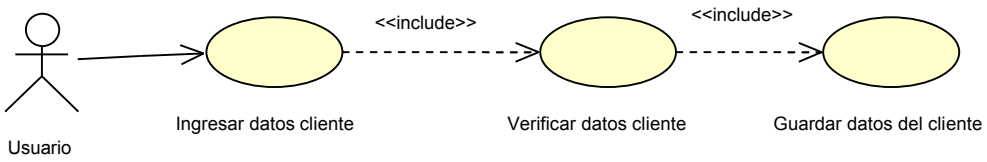

Figura 8. Error: casos de uso como DFD

Otro error frecuente es crear un caso de uso que es incluido por un solo caso de uso; por ejemplo, la figura 9 muestra el caso de uso "Buscar Cliente", el cual es incluido sólo por el caso de uso "Actualizar clientes". Se debe tener en cuenta que los casos de uso incluidos deben obtenerse luego de haber realizado las especificaciones de los casos de uso, ya que en ese momento es que se determinarán cuáles son los pasos que se repiten entre los diferentes casos de uso y es allí donde se determinan las relaciones de tipo "include".

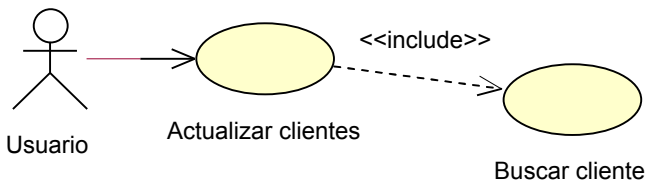

Figura 9. Error: el caso de uso que es incluido por uno solo.

Se puede encontrar bibliografía en la que se emplea la relación "use" entre casos de uso. Se debe tener en cuenta que dicha relación corresponde a versiones anteriores a UML versión 1.3, por lo que su utilización debe evitarse (ver figura 10).

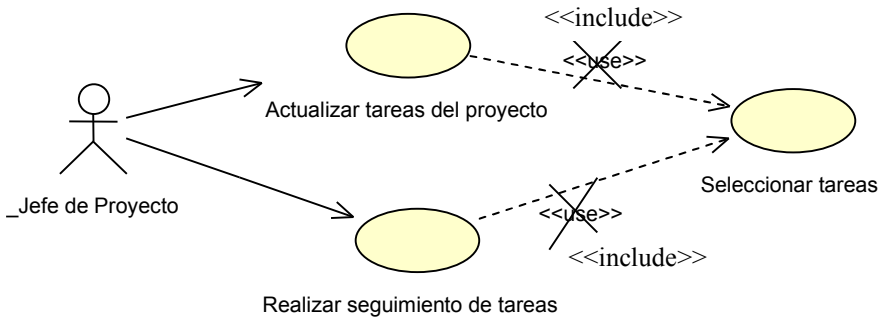

Figura 10. Error: uso de notación antigua de UML.

5. Conclusiones y Trabajo Futuro

En este artículo se muestran las actividades que se deben realizar para la especificación de requisitos utilizando casos de uso. Estas actividades han permitido minimizar los errores en la aplicación del estándar UML y lo que es importante, finalizar el proceso con un documento de especificación de requisitos libre de errores y útil para las etapas de análisis y diseño.

El uso de las relaciones de casos de uso es opcional y no es necesaria para realizar un documento de especificación de requisitos adecuado y detallado.

El presente trabajo es un inicio para el establecimiento de patrones en la aplicación de casos de uso, de manera similar a los patrones de diseño [4] y los antipatrones [2] de software.

6. Bibliografía

1. Bittner, K., *Why Use Cases Are Not Functions,* http://www.therationaledge.com, USA, 2000.
2. Brown, W.J., *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis,* John Wiley & Sons, USA, 1998.
3. Bruegge B., Allen, S., *Ingeniería de Software Orientado a Objetos*, Addison-Wesley, USA, 2002.
4. Gamma, E., *Design Patterns: elements of reusable object-oriented software*, Addison-Wesley, USA, 1995.
5. Heumann, J., *Introduction to Business Modeling Using the Unified Modeling Language*, The Rational Edge, March 2001, USA.
6. Larman, C., *UML y Patrones, Introducción al Análisis y Diseño Orientado a Objetos,* México, Prentice Hall Hispanoamericana, 1999
7. Ministerio de Administraciones Públicas, *Análisis del Sistema de Información-Métrica Versión 3,* España, 2000.
8. Object Management Group, *OMG Unified Modeling Language*, http://www.uml.org, USA, 1999
9. Rational Software, *Rational Unified Process version 2001A.04.00.13,* USA, 2001.
10. Schneider, G., Winters, J.P., *Applying Use Cases, Second Edition*, Addison-Wesley, Massachussets, USA, 2001.
11. Yourdon E., *Análisis Estructurado Moderno,* Prentice Hall Hispanoamericana, México, 1989.