# Acercamiento ontológico a la gestión del conocimiento en el mantenimiento del software

# Ontological approach to knowledge management in software maintenance

*Edgar Serna Montoya*[*]

Grupo de investigación de Ingeniería de Software. Universidad Nacional, Facultad de Minas. Transversal 51A 67B 90, Piso 5. Medellín, Colombia.

(Recibido el 20 de octubre. Aceptado el 6 de abril de 2010)

**Resumen**

Muchos documentos describen diseños ontológicos, pero pocos explican cómo se puede diseñar una ontología; además, muy pocos se enfocan en la aplicación de técnicas de gestión del conocimiento para el mantenimiento del software. En este trabajo se hace un análisis de varias de las ontologías propuestas para responder a esta necesidad, con la finalidad de que algunas de sus ideas puedan ayudar a las organizaciones de desarrollo de software en trabajos similares, y se describe una metodología para estructurar una ontología que, además del área del mantenimiento del software, se pueda aplicar a otras áreas del conocimiento.

----- *Palabras clave:* mantenimiento del software, ontología, gestión del conocimiento

**Abstract**

Many papers describe ontological designs but few explain how to design ontology. Moreover, very few focus on applying knowledge management techniques for software maintenance. This paper provides an analysis of several of the ontology's proposed to address this need, in order that some of their ideas can help software development organizations in similar work, and describes a methodology for structuring an ontology that, besides the area of software maintenance, can be applied in other areas of knowledge.

---- *Keywords:* software maintenance, ontology, knowledge management

* Autor de correspondencia: teléfono: + 57 + 4 + 448 76 66 Ext. 9611, fax: : + 57 + 4 + 384 97 97, correo electrónico: eserna@gmail.com (E. Serna)

## Introducción

Para realizar trabajos eficientes en los modelos de Ingeniería de Software y aplicarlos en cada una de sus fases, el conocimiento requerido es variado, de grandes proporciones y en incremento constante. La investigación en Ingeniería de Software se orienta hacia la gestión del conocimiento, procurando tomar las mejores decisiones y proporcionando a las organizaciones la información que requieren para el desarrollo de estas fases [1]. La fase de mantenimiento del software es una actividad en la que el conocimiento juega un importante rol; el nivel de conocimiento de quienes la realizan es complejo, voluminoso e intensivo en áreas como: el dominio del programa, de la organización, el pasado y presente de las prácticas de Ingeniería de Software, los lenguajes de programación, las metodologías de programación, las relaciones entre los módulos, las herramientas necesarias, entre otras [2]. Con frecuencia la información necesaria no se encuentra o es muy difícil de localizar y reconstruir, por lo que los probadores deben consultar la escasa documentación disponible o a los compañeros de trabajo, lo que origina que parte del conocimiento existente en los grupos de mantenimiento se pierda o no se utilice [3].

La gestión del conocimiento provee técnicas y métodos que ayudan a reducir la pérdida o el desaprovechamiento de conocimiento, y permite que los encargados del mantenimiento de software lo compartan [4]; a las organizaciones de desarrollo y mantenimiento de software les asegura beneficios como mejorar la calidad de productos y procesos, y reducción de costos y errores [5]. Sin embargo, antes de iniciar procesos de desarrollo de sistemas de este tipo, es importante identificar el conocimiento a gestionar, el lugar en el que se almacena y el lugar en que se requiere; además, dado que generalmente la organización no sabe cómo localizarlo o qué persona lo posee, esto también se convierte en tarea por realizar [6].

Para ayudar en esa gestión, los investigadores trabajan desde hace algún tiempo en la conceptualización de ontologías como modelos del dominio, que emergen como instrumento apropiado para la gestión del conocimiento. En relación con el mantenimiento y considerando los tipos en las ontologías que lo apoyan, los desarrollos ontológicos se nutren de experiencias y aportes diversos, pero pocos llegan a modelar e implementar una ontología representativa de esta área. Este artículo, un documento de revisión acerca de la temática, se estructura de la siguiente forma: el mantenimiento del software; las ontologías; análisis de algunas ontológicas alrededor del mantenimiento del software; descripción de una propuesta metodológica para gestionar el conocimiento en el mantenimiento del software y conclusiones y futuros trabajos que den continuidad a esta investigación.

### *Mantenimiento del software*

El mantenimiento de software se define como "cualquier modificación de un producto de software, después de su entrega, para corregir errores, mejorar el rendimiento u otros atributos, o a la acción de adaptar el producto a un entorno que cambia" [7]; "los cambios en la gestión de productos de software para mantenerlos actualizados y en pleno funcionamiento" [8]. Este tema, de mucha pertinencia e importancia en la Ingeniería de Software, recibe relativamente poca atención en la literatura técnica [9].

La planificación de mantenimiento de software debe comenzar con la decisión de desarrollar un nuevo sistema, y debe considerar los objetivos de calidad que IEEE recomienda [10]. Las organizaciones deben mantenerse al ritmo de estos cambios, lo que a menudo significa que deben modificar el software sobre el que apoyan sus actividades. Una solución factible para romper este círculo vicioso, es el desarrollo de sistemas de gestión de conocimiento para el mantenimiento del software y evitar una práctica común a la mayoría de las organizaciones: intentar volver a documentar sus sistemas de software, que es una operación costosa y en la que se alejan del beneficio de los programas instalados para centrarse en la documentación.

### Las ontologías

En la actualidad el término "*ontología*" suscita gran interés, especialmente desde que el *World Wide Web Consortium* -W3C- la considerara como la tecnología llamada a facilitar la infraestructura de conocimiento a la naciente Web Semántica o Web 3.0 [11], [12]. Para ser una tecnología, cuyo objetivo es clarificar, explicitar y consensuar el conocimiento relacionado con un dominio determinado, es paradójico que no alcance un consenso que ofrezca claridad acerca de lo que es o debería ser [13]. Lacy [14] manifiesta que el término ontología se sobrecarga, en tanto que su significado se refiere a diferentes cosas según quién lo defina. Una de las definiciones más citada en el ámbito de la ingeniería del conocimiento la define como "una especificación explícita de una conceptualización" [15].

"Una ontología es la descripción conceptual y terminológica de un conocimiento compartido acerca de un dominio específico. Dejando de lado la formalización e interoperabilidad de aplicaciones, esto no es más que la principal competencia del término: hacer mejoras en la comunicación utilizando un mismo sistema en lo terminológico y conceptual" [16]. De estas concepciones es importante tener en cuenta que, contrario a la filosófica –Ontología-, "ontología" debe considerarse no como una entidad natural que se descubre, sino como un recurso artificial que se crea con un objetivo determinado y para una aplicación concreta [17].

Por todo esto, diseñar una ontología implica hacer elecciones y seleccionar criterios y, ya que su objetivo es servir como referencia a personas, aplicaciones y organizaciones, esas elecciones y categorizaciones se deben consensuar y reconocer [18]. En el caso de los proyectos de mantenimiento de software es muy útil tener ontologías definidas acerca de la temática de gestión de los mismos, ya que se resuelven estos equívocos y evita las discusiones originadas en la comprensión del concepto mismo de la "petición de mantenimiento" [19].

### *Ontologías alrededor del mantenimiento del software*

Antes de estructurar un sistema de gestión del conocimiento para el mantenimiento del software, debe pensarse en modelar, estructurar y generalizar la información que se genera y consulta en dicho proceso. Para lograr eficientemente esta actividad se utilizan las ontologías con las que, según Gruber [20], se hace la especificación explícita de una conceptualización. Debido a que es una rama del conocimiento en permanente construcción y desarrollo, diferentes autores plantean sus ontologías en relación con el mantenimiento del software.

### *La ontología informal para el mantenimiento del software [21]*

Describe los aspectos más importantes para realizar estudios empíricos del mantenimiento del software; aunque tiene como objetivo particular identificar los factores de dominio que influyen en los resultados de estos estudios, propone otros que influyen en esta fase y sirven para modelar una ontología de mantenimiento. Se estructura en cuatro sub-ontologías: 1) De productos, en la que se definen los productos software que se van a mantener, su estructura interna, composición y versiones existentes; 2) De actividades, que define las actividades y recursos, dos tipos de elementos básicos en la gestión de un proyecto de mantenimiento; 3) De procesos de la organización, que define cómo llevar a cabo las actividades y cómo organizar el proceso de mantenimiento como tal; y 4) De agentes, que abarca la jerarquía de tipos de agentes existentes en la gestión del proyecto.

Aunque sirve como referencia para otras ontologías, algunas de las cuales se comentan en este trabajo, no llega a ser un modelo que direccione una aplicación ontológica de mantenimiento de software, y no tiene en cuenta la gestión del conocimiento en un contexto en el que sea posible aprovecharlo eficientemente por los probadores.

### *El enfoque orientado al concepto [22]*

Esta ontología parte de la definición abstracta de Gruber [20], de que es una especificación explícita de una conceptualización; refuerza la tesis de que representa un cierto punto de vista sobre una solicitud de dominio, en el que hay que definir los conceptos que viven en él de modo explícito y sin ambigüedades. Complementa la definición de Gruber al tener en cuenta el papel que los conceptos de la especificación explícita desempeñan. Su objetivo es considerar un enfoque ligero en el que es deseable, pero no necesario, alcanzar una serie de conceptos que se puedan usar como estándar para una comunidad más grande, y no la de crear ontologías formales y rigurosas para aplicar sobre dominios sometidos a consideración. Objetivo que refuerza al definir que una de las principales actividades en el desarrollo de software es obtener conocimiento y entendimiento acerca del dominio de la aplicación.

No tiene en cuenta que mucho conocimiento continúa implícito en los objetos resultantes: los vínculos entre los objetos, el conocimiento que se pierde con las mejoras iterativas o el conocimiento que las partes consideran de sentido común.

### *La ontología basada en el conocimiento [23]*

Su principio es que el desarrollo y el mantenimiento de software deben ser tareas de conocimiento intensivo, en las que se necesita conocer el dominio de la aplicación, el problema que soluciona el sistema, los requisitos del problema, la arquitectura del sistema, la forma como encajan las diferentes partes y la interacción del sistema con el medio ambiente. Se estructura en cinco aspectos: toma los cuatro de Kitchenham [21] y le agrega el del conocimiento relacionado con el dominio de aplicación, que utiliza para determinar el conocimiento que necesitan los encargados de realizar mantenimiento del software. Buena parte de sus actividades y aplicaciones se desarrolla en posteriores trabajos, [24], [25], que refrendan el concepto de desarrollo y mantenimiento de software como la comprensión de las necesidades de los usuarios y su mundo, y como un proceso en el que se convierte el código en una serie de decisiones de diseño.

No es claro en esta propuesta cómo definir un mapa de conocimiento alrededor del mantenimiento del software, ya que el tiempo para capturar y analizar la información es un factor crítico que no especifica claramente.

### *MANTIS: entorno para el mantenimiento integral del software [26]*

Es un grupo de ontologías que tiene como base la propuesta de Gruninger y Lee [27], aunque sólo se orienta a dos de los tres usos que éstos proponen: la comunicación y la reutilización y organización de conocimiento. El tercer uso, la inferencia computacional, no se incluye en la actual MANTIS, se propone como una de las líneas de trabajo futuras. Al mantener limitado el uso de la ontología, no es necesario formalizarla completamente, en su lugar, la representación de la información se logra mediante modelos y metamodelos. Esta propuesta no implica sólo aspectos de gestión, también integra, desde una perspectiva más amplia del proceso del negocio, la dimensión de gestión con la dimensión de ingeniería de software.

Su componente ontológico es uno de los más completos de los referenciados en esta investigación, ya que tiene en cuenta la fase de mantenimiento de software como un componente integral del proceso de la ingeniería de software y, porque mediante el uso de modelos y metamodelos, se acerca a conceptos como el OO y de aspectos.

### *La ontología basada en la reutilización de la información [28]*

Esta ontología parte de la premisa de que es conveniente para toda organización que la información y el conocimiento se procesen y almacenen de forma tal que se puedan reutilizar, y que para el mantenimiento es importante realizar una buena gestión de los mismos, ya

que proceden de distintas fuentes y etapas del ciclo de vida. Describe la manera de definir los conceptos involucrados en el mantenimiento del software y cómo representarlos en una ontología, potencializando el reuso de la información mediante técnicas de razonamiento basado en casos, de tal forma que los encargados del mantenimiento aprovechen las experiencias y lecciones aprendidas por otros. Tiene su origen en la ontología de Kitchenham y su grupo [21], al identificar los factores que influyen en el mantenimiento, pero además, indica un conjunto de aspectos dinámicos del dominio descrito en términos de estados, eventos y procesos. Su objetivo es definir una ontología con un nivel de abstracción más complejo, pero pierde de vista el concepto dinámico del producto de la ingeniería de software: el sistema.

Aunque su idea es reutilizar los conocimientos alcanzados en proyectos anteriores como ayuda para realizar los futuros, no se encuentra una clara y exhaustiva definición de cómo hacer que esa reutilización sea útil en la realización del mantenimiento del software.

### *La ontología de conceptos de ingeniería de software [29]*

La idea central de esta ontología es separar los conocimientos de ingeniería de software del dominio, de los conocimientos sobre las operaciones, los componentes de software y el sistema de metadatos. Además, hacer supuestos explícitos sobre el dominio, con lo que es posible hallar lagunas en el conocimiento acerca de la forma como se estructuran algunos lenguajes de programación OO [30]. Describe la relación entre los componentes de un sistema OO y las pruebas de software, las métricas y los requisitos, definidos como los distintos componentes de software. Asocia las métricas y las pruebas con los componentes del software y los requisitos con sus múltiples componentes.

El momento clave para utilizar esta ontología es al capturar información de los cambios, ya que se realiza mediante un objeto de propiedad que puede utilizarse en cualquier componente de software, en pruebas, métricas o requisitos, y denota cuándo se modificó por última vez. Es de esperar que este tipo de ontologías, con la codificación de características comunes del dominio de la ingeniería de software, tenga un alto componente de reutilización.

### *La ontología basada en el conocimiento del sistema [31]*

Busca representar los aspectos estáticos y dinámicos del proceso de mantenimiento del software desde el punto de vista de los procesos de negocio. Establece una relación entre la ontología y el conjunto de mejores prácticas de CMMI; relación en la que especificar la transferencia de conocimientos mediante las buenas prácticas que describen los modelos de madurez es difícil.

Aunque describe el proyecto de modelado del mantenimiento con base en la metodología de van Heijst [32], que representa el conocimiento de sentido común como reutilizable a través de dominios, la ontología resultante es un modelo de tareas sobre el concepto del formalismo de la ontología de dominio, la cartografía de la ontología en el papel del conocimiento de la tarea, y un modelo resultante que instancia la aplicación del dominio específico del conocimiento.

No tiene en cuenta muchos de los conceptos que participan en la solución, ya que limita su número al mostrar su utilidad en el mantenimiento del software de dominio. Debido a esto es necesario encontrar una forma de vincular esta arquitectura con una ontología de los conceptos de mantenimiento, además de analizar las tareas necesarias para construir un sistema de gestión de conocimiento que, quienes realizan la actividad, puedan utilizar de apoyo en su búsqueda de soluciones [33].

## ***Propuesta metodológica para diseñar ontologías***

Una ontología es la descripción conceptual y terminológica de un conocimiento compartido

acerca de un dominio específico y, aparte de la formalización e interoperabilidad de aplicaciones, la principal competencia del término es hacer mejoras en la comunicación utilizando un mismo sistema en lo terminológico y conceptual [34]. La metodología que se propone en este documento se basa en la propuesta de Noy y McGuinness [30], que definen todos los conceptos relevantes acerca de por qué desarrollar una ontología, y presentan una metodología para su diseño basada en los sistemas de representación del conocimiento declarativo. Y, aunque no existe una única forma ni metodología "*correcta*" para estructurar y desarrollar una ontología, a continuación se describen las fases generales que se deben considerar, así como los procedimientos posibles para diseñarla.

### *Determinar el dominio y el alcance*

Para determinar el dominio se hacen preguntas como: ¿Qué dominio cubrirá la ontología? ¿Qué uso se le dará? La información en ella, ¿a qué tipo de preguntas deberá responder? ¿Quién se encargará de usarla y de mantenerla? Las respuestas normalmente cambian en el proceso de estructuración de la ontología, pero la información que se recolecta cada vez sirve de ayuda para no exceder los límites y alcances del diseño. Para determinar el alcance se hacen preguntas de competencia que, como bosquejo, no requieren ser exhaustivas; además, la base de conocimientos de la ontología debería responderlas [35], y posteriormente se podrán utilizar para realizar las pruebas de control de calidad: ¿Contiene la ontología información suficiente para responder ese tipo de preguntas? ¿Qué nivel requieren las respuestas, detallado o representativo de un área en particular?

### *Considerar la reutilización de ontologías*

Muchas veces vale la pena considerar otros trabajados acerca de la temática, para verificar si es posible utilizarlos para refinar y extender la ontología que se diseña [36]. El reuso de ontologías puede hacerse cuando la que se estructura requiere interactuar con otras aplicaciones o cuando el nivel de formalismo en el que otras ontologías se expresan pasa a segundo plano; además, "traducir" una ontología desde un formalismo particular a otro no es una tarea complicada [37], [38]. En la Internet existen bibliotecas de ontologías que se pueden reutilizar:

- Biblioteca de Ontolingua [39]
- Guía de recursos sobre ontologías [40]
- La biblioteca de ontologías DAML [41]
- Biblioteca semántica WebQuest [42]

Igualmente, es posible adquirir un buen número de ontologías comerciales:

- UNSPSC [43]
- RosettaNet [44]
- DMOZ [45]

En cualquier caso, la conceptualización y elaboración de una ontología se debe realizar para cada una de las ontologías parciales definidas en el alcance y teniendo en cuenta [46]:

1. Definir el glosario de conceptos a partir de las fuentes de conocimiento citadas.
2. Definir las interrelaciones semánticas entre dichos conceptos representándolas mediante un diagrama de clases UML.
3. Analizar los conceptos relacionados para identificar las partes comunes a dos o más conceptos.
4. Identificar los atributos terminales de todos los conceptos.
5. Completar las tablas de atributos de conceptos e incluir los atributos no terminales.
6. Comprobar la completitud de todas las tablas de atributos e indicar si pertenece a la capa de descripción del artefacto, a su interfaz o al contexto.

### *Enumerar términos importantes para la ontología*

Realizar, de manera estructurada, una lista de todos los términos a enunciar y sus propiedades, pero sin tener en cuenta los conceptos que representan las relaciones entre ellos, las propiedades que puedan tener o si los conceptos son clases o slots, ya que, en esta metodología, se tienen en cuenta en las siguientes fases [47]; de lo que se trata es de crear las definiciones de los conceptos en la jerarquía para luego describir sus propiedades de forma sucesiva.

## *Definir las clases y la jerarquía de clases*

Existen varios enfoques para desarrollar jerarquías de clases [48]:

- *Top-down*, en el que primero se definen los conceptos más generales en el dominio y su respectiva especialización.
- *Bottom-up*, que comienza con la definición de clases específicas y luego genera las hojas de la jerarquía con su respectivo agrupamiento de clases en conceptos más generales.
- *Desarrollo combinado*, resulta de combinar los anteriores y comienza por definir los conceptos más importantes para luego generalizarlos y especializarlos apropiadamente.

Ninguno de los tres enfoques puede considerarse mejor a los otros, su escogencia depende en gran medida de la visión que se tiene del dominio: si es sistemática, lo lógico es utilizar el enfoque *top-down*; el enfoque combinado es el más fácil de aplicar, ya que los "*conceptos del medio*" generalmente son los conceptos más descriptivos en el dominio [49]; si visión es es pensar primero en una clasificación más general, podría funcionar mejor el enfoque *top-down;* si la visión es comenzar con un listado de ejemplos específicos, el enfoque *bottom-up* podría ser el más apropiado.

Sea cual sea el enfoque que se elija, la primera tarea es definir las clases: de la lista de términos generada al *Determinar el dominio y el alcance*, se seleccionan los que describen objetos con existencia independiente, que se convierten en las clases de la ontología y en la base de la jerarquía de clases. Luego se organizan en una taxonomía jerárquica teniendo en cuenta que una instancia de una clase puede ser instancia de otra clase; es decir, que la clase *B* representa un concepto que es un "*tipo de*" *A* [50].

### *Definir las propiedades de las clases*

Para responder a las preguntas de competencia halladas al *Determinar el dominio y el alcance*, las clases aisladas no ofrecen la suficiente información, por lo que es necesario describir la estructura interna de conceptos: se seleccionan clases de la lista de términos estructurada al *Enumerar los términos importantes para la ontología* y, muy probablemente, los términos restantes son sus propiedades; para cada propiedad en la lista se determina a qué clase describe, con lo que se convierten en los *slots* de la clase [20].

Además de las propiedades identificadas previamente, es necesario añadir los *slots* a la clase. Debe tener en cuenta que todas las subclases de una clase heredan los *slots* de la misma, y que un *slot* debe estar yuxtapuesto a la clase más general que puede tener esa propiedad [49].

### *Definir las facetas de los slots*

Los slots tienen diferentes facetas [49]:

- Cardinalidad, que define cuántos valores puede tener un slot.
- Tipo de valor, que describe qué tipos de valores puede contener: String, es una cadena de caracteres; Number, valores numéricos; Boolean, si/no, verdadero/falso; Enumerated, que los define el desarrollador; Instance, admiten la definición de relaciones entre individuos.
- Dominio y rango, cuyas reglas de determinación son: al definir un dominio

o rango de un slot, se debe encontrar la o las clases más generales, que puedan ser el dominio o rango de los slots; no se debe definir un dominio o rango que sea muy general: todas las clases en el dominio deben ser descritas por el slot, y las instancias de las clases en el rango de un slot deben ser rellenos potenciales del slot [48].

*Crear instancias y cardinalidades*

El proceso es el siguiente: 1) elegir la clase, 2) crear la instancia individual de la clase y 3) llenar los valores del slot [51], [52]. Se debe tener en cuenta: 1) decidir si un concepto particular será una clase o una instancia individual en la ontología depende su aplicación [52], [54]; 2) para decidir dónde terminan las clases y comienzan las instancias, es necesario hallar primero el nivel más bajo de granularidad en la representación, que también lo determina la aplicación potencial de la ontología [55]; 3) sólo las clases se pueden representar en una jerarquía, ya que los sistemas de representación de conocimiento no tienen la noción de sub-instancias, por lo que, si los términos tienen una jerarquía natural, se deben definir como clases aunque no tengan ninguna instancia propia [56], [57].

## Conclusiones y trabajo futuro

- La gestión del conocimiento es una técnica muy importante para facilitar el trabajo de los encargados del mantenimiento del software, por lo que es necesario contar con un método para detectar sus fuentes y ubicación en la organización; una manera de hacerlo, rápida y eficientemente, es utilizar agentes inteligentes mediante ontologías.
- El diseño ontológico es un proceso que exige creatividad, por lo que las ontologías nunca serán iguales aunque se estructuren sobre el mismo dominio. La potencial aplicación de la ontología, así como la comprensión y aspecto del dominio por parte del diseñador, afectan las opciones del diseño mismo de la ontología.
- No se sabe si algo es bueno hasta que se lo pone a prueba, se puede evaluar la calidad de la ontología solamente utilizándola en las aplicaciones para las cuales se diseñó. Recordar que no existe una sola forma para estructurar ontologías, que pueda considerarse correcta.

En cuanto al trabajo futuro debe considerarse:

- Plantear la creación de una ontología en una herramienta automatizada para probar la metodología propuesta, y extenderla con algún lenguaje de definición de reglas para superar las limitaciones del lenguaje natural en cuanto a la composición de los escenarios. Se recomienda elegir uno que se ajuste a las necesidades de la implementación, ya que el objetivo final es encontrar la manera de definir reglas que faciliten el entendimiento entre los sistemas y las personas.
- Mejorar los procesos de gestión del conocimiento, ya que en esta propuesta se descarta el conocimiento inconsistente al mantenimiento del software y, debido a que la misma ontología puede ser parcialmente inconsistente, podría mejorarse si fuera posible descartar únicamente las partes inconsistentes en vez de toda la ontología. Para lograr esto, puede ser que al modificar las ontologías fuente y eliminar de ellas el contenido inconsistente, se logre una ontología final más consistente.

## Referencias


1. I. Lindvallv. “Knowledge management in software engineering”. *IEEE Software*. Vol. 19. 2002. pp. 26-38.
2. T. M. Pigoski. *Practical software maintenance: best practices for managing your software investment*. Ed. John Wiley & Sons. New York. 1996. pp. 300-303.
3. D. B. Walz, J. J. Elam, B. Curtis. “Inside a software design team: knowledge acquisition, sharing, and integration”. *Communications of the ACM*. Vol. 36. 1993. pp. 63-77.
4. O. M. Rodríguez, A. I. Martínez, J. Favela, A. Vizcaíno, M. Piattini. “Understanding and supporting knowledge flows in a community of software developers”. *Lecture Notes in Computer Science*. Vol. 3198. 2004. pp. 52-66.

5. T. Dingsøyr, R. Conradi. "A survey of case studies of the use of knowledge management in software engineering". *International Journal of Software Engineering and Knowledge Engineering*. Vol. 12. 2002. pp. 391-414.

6. J. Nebus. "Framing the knowledge search problem: whom do we contact, and why do we contact them?". *Academy of Management Best Papers Proceedings*. 2001. pp. h1-h7.

7. S. Mamone. "The IEEE standard for software maintenance". *ACM SIGSOFT Software Engineering Notes*. Vol. 19. 1994. pp. 75-76.

8. R. Singh. "ISO/IEC draft international standard 12207, software life-cycle processes". *IFIP Transactions*. Vol. A-55. 1994. pp. 111-119.

9. R. S. Pressmann. *Software engineering: a practitioner's approach*. McGraw-Hill. México. 2005. pp. 807-812.

10. Software Engineering Standards Committee. "IEEE Standard for a software quality metrics methodology, Std. 1061-1998". *Technical Report*. 1998. pp. 24-26.

11. L. Lefort, K. Taylor, D. Ratcliffe. "Towards scalable ontology engineering patterns: lessons learned from an experiment based on W3C's part-whole guidelines". *Proceedings of the second Australasian workshop on Advances in ontologies*. Hobart (Australia). Vol. 72. 2006. pp. 31-40.

12. I. Horrocks. "Ontologies and the semantic web". *Communications of the ACM*. Vol. 51. 2008. pp. 58-67.

13. J. A. Evans. "Electronic Publication and the narrowing of science and scholarship". *Science*. Vol. 321. 2008. pp. 395-399.

14. L. W. Lacy. *OWL: Representing information using the Web ontology language*. Ed. Trafford Publishing. Bloomington (USA). 2005. pp. 300-302.

15. T. R. Gruber. "Towards principles for the design of ontologies used for knowledge sharing". *International Journal of Human-Computer Studies.* Vol. 43. 1995. pp 907-928.

16. M. Reuver, T. Haaker. "Designing viable business models for context-aware mobile services". *Telematics and Informatics*. Vol. 26. 2009. pp. 240-248.

17. K. Mahesh. *Ontology development for machine translation: ideology and methodology*. Computing Research Laboratory. Technical Report MCCS-96-292. New México State University. Las Cruces (NM). 1996. pp. 5-6.

18. K. M. Oliveira, N. Anquetil, K. de Sousa, M. G. Batista. "Knowledge for software maintenance". *Fifteenth International Conference on Software Engineering and Knowledge Engineering.* San Francisco (CA). 2003. pp. 61-68.

19. F. G. Ruiz, A. Vizcaíno, M. Piattini, F. García. "An Ontology for the management of software maintenance projects". *International Journal of Software Engineering and Knowledge Engineering*. Vol. 14. 2004. pp. 323-349.

20. T. R. Gruber. "A translation approach to portable ontology specifications". *Knowledge Acquisition*. Vol. 5. 1993. pp. 192-220.

21. B. A. Kitchenham, G. H. Travassos, A. von Mayrhauser, F. Niessink, N. F. Schneidewind, J. Singer, S. Takada, R. Vehvilainen, H. Yang. "Towards ontology of software maintenance". *Journal of Software Maintenance: Research and Practice*. Vol. 11. 1999. pp. 365-389.

22. D. Deridder. *Facilitating software maintenance and reuse activities with a concept-oriented approach*. Technical report. Vrije Universiteit Brussel. Belgium. 2002. pp. 2-3.

23. K. M. Oliveira, N. Anquetil, K. de Sousa, M. G. Batista. "Organizing the knowledge used in software maintenance". *Journal of Universal Computer Science*. Vol. 9. 2003. pp. 641-658.

24. K. M. Oliveira, N. Anquetil, K. de Sousa, M. G. Batista. "Legacy software evaluation model for outsourced maintainer". *Software Maintenance and Reengineering*. Eighth European Conference on CSMR'04. Tampere (Finlandia). 2004. pp. 65-72.

25. K. M. Oliveira, N. Anquetil, K. de Sousa, M. G. Batista. "Software maintenance seen as a knowledge management issue". *Information and Software Technology*. Vol. 49. 2007. pp. 515-529.

26. F. G. Ruiz. *MANTIS: Entorno para el Mantenimiento Integral del Software*. Tesis doctoral. Universidad de Castilla-La Mancha. 2003. pp. 45-50.

27. M. Gruninger, J. Lee. "Ontology applications and design". *Communications of the ACM*. Vol. 45. 2002. pp. 39-41.

28. A. Vizcaíno, J. P. Soto, F. García, F. Ruiz, M. Piattini. "Aplicando gestión del conocimiento en el proceso de mantenimiento del software". *Revista Iberoamericana de Inteligencia Artificial*. Vol. 10. 2006. pp. 91-98.

29. D. Hyland-Wood, D. Carrington, S. Kaplan. "Enhancing software maintenance by using semantic web techniques". *5th International Semantic Web Conference*. Athens (USA). 2006. pp. 2-4.

30. N. Noy, D. Mcguinness. *Ontology development 101: a guide to creating your first ontology.* Technical Report

KSL-01-05. Stanford Knowledge Systems Laboratory, Stanford University. Palo Alto (CA). 2001. pp. 3-4.

31. A. April, J-M Desharnais, R. A. Dumke. "A formalism of ontology to support a software maintenance knowledge-based system". *Proceedings of the Eighteenth International Conference on Software Engineering & Knowledge Engineering Conference*. San Francisco (CA). 2006. pp. 331-336.
32. G. van Heijst, A. Schreiber, B. Wielinga. "Using explicit ontologies in KBS development". *International Journal of Human-Computer Studies*. Vol. 46. 1996. pp. 2-3.
33. B. Sarder, S. Ferreira. "Developing systems engineering ontologies". *System of Systems Engineering, SoSE '07. IEEE International Conference.* San Antonio, USA. 2007. pp. 1-6.
34. J. M. Park, J. H. Nam, Q. P. Hu, H. W. Suh. "Product ontology construction from engineering documents". *International Conference on Smart Manufacturing Application, ICSMA'08*. Goyang-si (South Korea). 2008. pp. 305-310.
35. M. Gruninger, M. S. Fox. "Methodology for the design and evaluation of ontologies". *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing*. Montreal. 1995. pp. 73-83.
36. A. Gómez-Pérez. "Knowledge sharing and reuse". *The Handbook of Applied Expert Systems*. J. Liebowitz (editor). Ed. CRC Press. Boca Raton (USA). 1998. pp. 10-1-10.36.
37. M. A. Musen. "Dimensions of knowledge sharing and reuse". *Computers and Biomedical*. Vol. 25. 1992. pp. 435-467.
38. T. R. Rothenfluh, J. H. Gennari, H. Eriksson, A. R. Puerta, S. W. Tu, M. A. Musen. "Reusable ontologies, knowledge-acquisition tools, and performance systems: PROTEGE-II solutions to Sisyphus-2". *International Journal of Human-Computer Studies*. Vol. 44. 1996. pp. 303-332.
39. http://www.ksl.stanford.edu/software/ontolingua/. Consultada el 12 de marzo de 2009.
40. http://es.geocities.com/ontologias_y_tesauros/guia_de_recursos_sobre_ontologias.htm. Consultada el 23 de enero de 2009.
41. http://www.daml.org/ontologies/. Consultada el 1 de marzo de 2009.
42. http://cfievalladolid2.net/webquest/common/index.php. Consultada el 20 de mayo de 2009.
43. www.unspsc.org. Consultada el 20 de mayo 2009.
44. www.rosettanet.org. Consultada el 14 de marzo de 2009.
45. www.dmoz.org . Consultada el 23 de febreo de 2009.
46. C. Tautz, C. G. von Wangenheim. *REFSENO: A representation formalism for software engineering ontologies*. Technical Report IESE-Report No. 015.98/E. Fraunhofer Institute for Experimental Software Engineering. Kaiserslautern (Germany). 1999. pp. 61-71.
47. A. Borgida, R. J. Brachman, D. L. McGuinness, L. A. Resnick. "CLASSIC: a structural data model for objects". *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*. Portland. 1998. pp. 59-67.
48. M. Uschold, M. Gruninger. "Ontologies: principles, methods and applications". *Knowledge Engineering Review*. Vol. 11. 1996. pp. 93-155.
49. E. Rosch. "Principles of categorization". *Concepts: core readings*. E. Margolis, S.Laurence (editors). Ed. MIT Press. Massachusetts (USA). 1999. pp. 189-206.
50. H. Ning, D. Shihan. "Structure-based ontology evaluation". *e-Business Engineering, ICEBE'06*. IEEE International Conference. Shanghai. 2006. pp. 132-137.
51. D. L. McGuinness, J. Wright. "An industrial strength description logic-based configurator platform". *IEEE Intelligent Systems*. Vol. 13. 1998. pp. 69-77.
52. K-S Choi. "IT ontology and semantic technology". *Natural Language Processing and Knowledge Engineering, NLP-KE'07*. International Conference. Beijing. 2007. pp. 14-15.
53. M. Fernández, A. Gómez-Pérez, N. Juristo. "Methontology: from ontological art towards ontological engineering". *AAAI Spring Symposium*. University of Stanford. Palo Alto (CA). 2007. pp. 33-40.
54. R. F. García, M. Piattini. *Calidad en el desarrollo y mantenimiento del software*. Ed. Rama. Madrid. 2003. pp. 240-244.
55. Software Engineering Standards Committee of IEEE Computer Society. *STD 1074-1997: IEEE Standard for developing software life cycle processes*. Technical Report. Washington. 1997. pp. 96.
56. F. G. Ruiz, C. Calero, M. Piattini. *Ontologies for software engineering and software technology*. Ed. Springer. London. 2006. pp. 339-345.
57. L. Zhang, S. Xia, Y. Zhou, A. Xia. "User defined ontology change and its optimization". *Chinese Control and Decision Conference, CCDC'08*. Yantai. Shandong. 2008. pp. 3586-3590.