Aplicando Gestión del Conocimiento en el Proceso de Mantenimiento del Software

Aurora Vizcaíno, Juan Pablo Soto, Félix García, Francisco Ruiz, Mario Piattini

Grupo ALARCOS
Departamento de Tecnologías y Sistemas de Información
Centro Mixto de Investigación y Desarrollo de Software UCLM-Soluziona
Universidad de Castilla-La Mancha
Paseo de la Universidad, 4, 13071 Ciudad Real (España)
{Aurora.Vizcaino | Felix.Garcia | Francisco.RuizG | Mario.Piattini }@uclm.es
jpsoto@proyectos.inf-cr.uclm.es

Resumen

En toda organización es conveniente que la información y el conocimiento se procesen y almacenen de forma que estos se puedan reutilizar. En el caso del mantenimiento del software es todavía más importante realizar una buena gestión de la información y del conocimiento ya que estos provienen de distintas fuentes y etapas del ciclo de vida. Sin embargo, en la actualidad existen muy pocos trabajos enfocados en la aplicación de técnicas de gestión del conocimiento en el mantenimiento del software. En este artículo se describe cómo hemos definido los conceptos involucrados en el mantenimiento del software y cómo se han representado en una ontología, la cual posteriormente ha sido implementada en un sistema de gestión del conocimiento usando REFSENO. Todo ello con el fin de potenciar la reutilización de la información (usando técnicas de razonamiento basado en casos), de forma que los ingenieros de mantenimiento puedan aprovechar la experiencia y lecciones aprendidas de otros trabajadores.

Palabras clave: Mantenimiento del software, gestión del conocimiento, ontologías, razonamiento basado en casos, REFSENO.

1. Introducción

Los procesos de desarrollo y mantenimiento del software generan diariamente gran cantidad de información [Rus&Lindvall02]. Sin embargo, esta información raramente se gestiona de forma que se conozca quién la ha generado, dónde se va a almacenar y a quién le puede resultar útil, lo cual implica que con frecuencia se "reinvente la rueda" y no se aproveche el "capital intelectual" de una organización. Este hecho es aún más grave en el caso del mantenimiento, el cual es considerado el proceso más costoso (en cuanto a dinero y esfuerzo se refiere) y más largo del ciclo de vida del software [Card90; Pigoski97; Polo et al. 99].

Durante esta etapa la información no sólo proviene de los profesionales involucrados en las actividades de mantenimiento, sino también del propio producto que se mantiene, de las causas que motivaron el mantenimiento, de los clientes y usuarios, así como de los procesos, metodologías y herramientas utilizadas por la organización. Además, la información cambia constantemente debido a la propia naturaleza del mantenimiento, que surge, entre otras razones, de la necesidad de adaptar sistemas software a un entorno que a su vez está siempre evolucionando [Oliveira et al. 03]. Por la misma razón el conocimiento que un ingeniero de

mantenimiento posee también evoluciona. Por ejemplo durante la fase de "desarrollo inicial" el grupo de mantenimiento adquiere conocimiento acerca del dominio de la aplicación, requisitos de usuario, roles, algoritmos, formato de los datos, capacidades y debilidades de la arquitectura del programa, etc. En la fase de "evolución del software" surgen nuevos usuarios y requisitos que propician nuevos cambios y producen nuevo conocimiento en los ingenieros [Bennett00], y de forma similar cada fase genera nueva **información** que es interiorizada [Nonaka&Takeuchi95] por los ingenieros, que a su vez le añaden su experiencia convirtiendo la información en **conocimiento** tácito. A partir de este momento en el resto del artículo, cuando la fuente sea un ingeniero le llamaremos conocimiento y cuando sea un documento u otro artefacto le llamaremos información.

Por otra parte, cada ingeniero de mantenimiento posee un conocimiento que puede ser útil para los demás miembros del equipo. Por ejemplo, cuando una persona modifica un método que afecta a otro que a su vez va a ser modificado por otra persona, la primera debería informar a la segunda. En consecuencia, debe existir cierta colaboración y comunicación entre los ingenieros del mantenimiento, siendo muy conveniente saber qué actividades está desarrollando cada empleado en cada momento. Otro aspecto a tener en cuenta en el proceso de mantenimiento es la experiencia que el empleado posee, ya que el factor que más influye en la productividad del mantenimiento es la presencia de por lo menos un miembro del equipo con experiencia en la aplicación que se está manteniendo [Banker et al. 91]. En [Banker et al. 02] se explica que los sistemas mantenidos por programadores sin experiencia suelen tener mayor número de errores en promedio. Esto demuestra la importancia que la experiencia tiene en el proceso de mantenimiento. Por ello, es útil capturar, en la medida de lo posible, parte de la experiencia que un empleado posee y almacenarla en un sistema de forma que esta pueda ser aprovechada por nuevos empleados o por otros compañeros.

Un problema adicional que complica el proceso de mantenimiento es la escasa documentación que suele existir relacionada con el software a mantener y la obsolescencia de la documentación asociada. Este hecho se acentúa en el caso de software heredado de otras organizaciones sin documentación que describa sus características [Sousa et al. 04].

Por todos estos motivos es necesario desarrollar técnicas y herramientas que ayuden a los ingenieros del mantenimiento a realizar sus tareas. Nuestra propuesta consiste en el desarrollo de un sistema de gestión de conocimiento capaz de procesar y almacenar la información generada durante el proceso del mantenimiento, con el fin de diseminarlo y reutilizarlo. Además, ello permitirá disminuir la probabilidad de repetir errores, incrementando la competitividad de la organización [de Looff97].

Como requisito previo a la construcción de un sistema de gestión de conocimiento es necesario modelar la información generada durante el proceso de mantenimiento del software y además estructurarla de forma que se facilite su reutilización. Con dicho fin hemos desarrollado un modelo del proceso de mantenimiento del software en el cual los objetos, conceptos, entidades y sus relaciones son representados explícitamente. Este modelo y su correspondiente ontología han sido implementados usando la metodología REFSENO (Representation Formalism for Software Engineering Ontologies). Una ventaja de REFSENO es que facilita la utilización de técnicas de razonamiento basado en casos, lo cual permite que el sistema que se ha desarrollado proponga soluciones a problemas parecidos a otros que han ocurrido anteriormente.

El resto de este artículo se ha estructurado de la siguiente forma: la sección 2 presenta una descripción de la ontología y su implementación con REFSENO. La sección 3 ilustra cómo la técnica del razonamiento basado en casos fue utilizada para reutilizar la información. Finalmente se presentan las conclusiones y los trabajos futuros.

2. Modelado e Implementación de una Ontología para el Mantenimiento del Software

Una ontología define de forma explicita y sin ambigüedad los conceptos de un dominio y sus relaciones [Deridder02]. Además, una ontología facilita la gestión de conocimiento [Maedche et al. 03], su compartición e integración [Falbo et al. 99].

La ontología que se describe en este artículo está basada en la ontología de [Kitchenham et al. 99], en la que se definen todos los conceptos relevantes para la clasificación de estudios empíricos en el mantenimiento del software. Dicha ontología está dividida en varias sub-ontologías:

- Sub-ontologia de productos: define los productos software que son mantenidos, su estructura y su evolución.
- Sub-ontologia de actividades: describe los distintos tipos de actividades para el mantenimiento del software.

- Sub-ontología de procesos: esta subontología está dividida en dos diferentes enfoques, definiéndose una subontología para cada uno de ellos:
  1. Sub-ontología de procedimientos: define cómo aplicar métodos, técnicas y herramientas a las actividades y cómo los recursos son utilizados para poder llevar a cabo las actividades
  2. Sub-ontología de organización del proceso: esta sub-ontología se enfoca en dar soporte a los procesos organizacionales relacionados con las actividades del mantenimiento del software, y cómo el ingeniero de mantenimiento debe organizarse, así como las obligaciones que éste contrae.
- Sub-ontología de agentes: describe las habilidades y roles necesarios para llevar a cabo una actividad, qué tipo de responsabilidades tiene cada persona, y cómo los actores que intervienen en los procesos de la organización (ingeniero de mantenimiento, cliente y usuario) se relacionan.

Otros autores también han basado sus propuestas en la ontología de Kitchenham, como la de Oliveira, que presentan una ontología para determinar el conocimiento que es necesario para los encargados del mantenimiento del software. En su ontología se proponen cinco aspectos en lugar de los cuatro descritos anteriormente, sin embargo, cuatro de ellos son similares a las sub-ontologías anteriormente descritas. Los aspectos considerados en [Oliveira et al. 03] son: el conocimiento acerca del sistema que corresponde a la sub-ontología del producto, el conocimiento relacionado con las habilidades del ingeniero de mantenimiento que corresponde a la sub-ontología de agentes; el conocimiento acerca de la actividad del mantenimiento la cual corresponde a la sub-ontología de actividades, el conocimiento acerca de la estructura de la organización que corresponde a la sub-ontología de procesos de la organización y el conocimiento relacionado con el dominio de aplicación, el cual no ha sido considerado por la ontología de Kitchenham.

[Deridder02] presenta una ontología orientada a conceptos, es decir, se enfoca en los conceptos que son reutilizados, los cuales, por otro lado, dependen de cada tipo de organización.

Sin embargo, nuestro objetivo es definir una ontología con un nivel de abstracción más elevado, enfocado en la gestión de proyectos de mantenimiento del software con un punto de vista de procesos de negocio. La ontología desarrollada para tal fin se describe en la siguiente sección.

2.1 Ontología del mantenimiento

Este nombre es una simplificación ya que, realmente, se refiere a la ontología de la gestión de proyectos de mantenimiento de software. Para poder gestionar un proyecto de mantenimiento de software, es necesario identificar los factores que influyen en el mantenimiento, tal y como lo hace la ontología de Kitchenham. Pero además, es necesario indicar un conjunto de aspectos dinámicos del dominio, descritos en términos de estados, eventos, procesos, etc. Por ello, hemos extendido la ontología de Kitchenham especificando tanto aspectos estáticos como dinámicos (que por razones de claridad se han desglosado en tres ontologías y cuatro sub-ontologías). Los aspectos dinámicos son modelados mediante flujos de trabajo [Ruiz et al. 02].

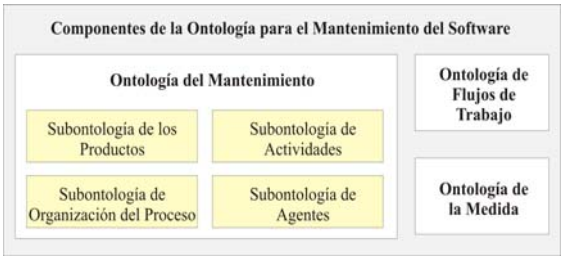

Figura 1. Estructura de la Ontología del Mantenimiento del Software

Para poder representar los aspectos estáticos, se definió una ontología llamada “Mantenimiento”, la cual se divide en cuatro subontologías (tal como muestra la Figura 1): La *sub-ontología de los productos*, la *sub-ontología de las actividades*, la *sub-ontología de organización del proceso* y una cuarta llamada *sub-ontología de agentes*. El número de ontologías estáticas coincide con las propuestas en [Kitchenham et al. 99]. No obstante, éstas han sido extendidas, añadiendo conceptos relacionados con la gestión de proyectos de mantenimiento, por ejemplo incorporando el concepto de “Versión” que no aparecía en ninguna de las sub-ontologías propuestas por Kitchenham ni por Oliveira pero es un concepto crítico para el mantenimiento del software.

La parte dinámica es representada por la ontología de los *flujos de trabajo*, la cual define los siguientes tres aspectos relevantes para el mantenimiento:

- Descomposición de actividades.
- Restricciones temporales entre las actividades (por ejemplo, el orden en el que deben realizarse las actividades).

- Control de las actividades y de la ejecución de los procesos durante el desarrollo de los proyectos.

Además, hemos diseñado una tercera ontología llamada *Ontología de la medida* que representa aspectos tanto dinámicos como estáticos relacionados con los conceptos de la medición.

Por cuestiones de espacio, en este trabajo sólo se presenta un ejemplo de cómo se ha modelado e implementado la ontología. Una descripción más detallada se puede encontrar en [Ruiz et al. 04]. En la Figura 2 se muestra, mediante el uso de diagramas de clases en UML el diagrama ontológico de la sub-ontología de los productos la cual hace énfasis en el componente producto ya que representa el concepto más importante. Esta sub-ontología define los productos software que son mantenidos, su estructura interna, composición y las versiones que existen de ellos. Se ha optado por usar diagramas UML porque coincidimos con la opinión de [Gómez-Pérez et al. 04] quienes consideran que UML es una buena técnica para modelar ontologías, ya que es fácil de usar para ingenieros del software y no resulta demasiado complejo de entender para personas ajenas al mundo de la informática. Además, existen numerosas herramientas CASE que facilitan la creación y modificación de estos diagramas.

Como se puede apreciar en la Figura 2, un producto software" puede tener diferentes versiones, las cuales están formadas por un conjunto de artefactos. Por ejemplo, para un producto llamado "Ventas" pueden existir diferentes versiones y cada versión puede estar compuesta de varios artefactos. El concepto versión tiene sus propios atributos, tales como número, fecha, etc., pero por cuestiones de simplicidad, no se han representado en el diagrama. El diagrama anterior sólo muestra una visión resumida de la ontología en cuestión.

2.2 Implementación de la Ontología

Con el fin de sistematizar la implementación de la ontología decidimos usar una metodología. En la literatura se proponen diferentes metodologías, por ejemplo, en [Hikita&Matsumoto01] se utiliza una representación basada en lógica de predicados de primer orden. Otros enfoques se basan en "frames" como Ontolingua [Gruber93], que es uno de los lenguajes ontológicos más utilizados.

Después de un estudio de las diferentes metodologías se optó por utilizar la metodología REFSENO (Representation Formalism for Software Engineering Ontologies) [Tautz98] debido a los siguientes motivos:

1. Como el propio nombre indica, fue especialmente diseñada para el desarrollo de ontologías en la ingeniería del software.
2. REFSENO hace una distinción entre los diferentes niveles de conocimiento: conceptual y específico del contexto, en cambio los enfoques citados anteriormente representan un nivel de abstracción más elevado.
3. La metodología propone diferentes técnicas para revisar la consistencia de la ontología y, además, tiene métodos para controlar la consistencia de las instancias en un nivel de implementación, característica que otras metodologías no consideran.
4. Facilita el uso de razonamiento basado en casos ya que considera el peso de cada atributo lo cual ayuda a calcular funciones de similitud.

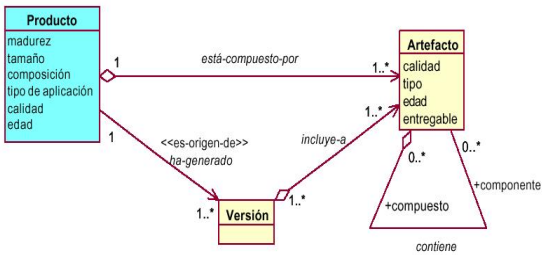

Figura 2. Diagrama de la subontología de los productos

El primer paso para implementar la ontología, según la metodología REFSENO, es definir un glosario de conceptos que contenga la descripción y propósito de los conceptos previamente representados en el diagrama de la sub-ontología. Cada concepto corresponde a una fila de la tabla y existe un glosario de conceptos por ontología y sub-ontología representada en la Figura 1. En este artículo se muestra, a modo de ejemplo, el glosario de conceptos de la sub-ontología de productos (Tabla 1).

El segundo paso es construir una tabla de atributos terminales para cada concepto definido en la tabla de glosarios. En la Tabla 2 se muestra la tabla del concepto Producto. Las tablas de los conceptos Artefacto y Versión se han omitido por razones de espacio.

Los atributos tienen un tipo determinado que bien puede ser uno de los que REFSENO propone o uno que el usuario define. En la Tabla 2 se muestran algunos de los tipos que hemos definido, por ejemplo "MaturityType", el cual define un rango de tipos de madurez que un producto software puede tener. El

valor del peso estándar (última columna de la tabla) representa el valor utilizado por las funciones de similitud (explicadas mas adelante). REFSENO distingue tres tipos de capas: artefacto, interfaz y contexto. Los atributos pueden pertenecer a cualquiera de estas capas. Los atributos de la capa artefacto (denotados en la tabla con la palabra artefacto) describen las características de la instancia. Los atributos de la capa de interfaz (denotado con I/F) caracterizan cómo una instancia en particular puede ser integrada en el sistema. Por último, los atributos de la capa de contexto definen el entorno en que la instancia se va a usar.

En la Tabla 2 solamente hay un atributo de la capa de interfaz el cual indica en qué artefactos se puede descomponer el producto. Obviamente, si el producto tiene una relación "Has-descomposition" con el concepto "artefacto", este componente debe tener la relación opuesta "Descomposition-of" indicando el producto del cual el artefacto forma parte.

Concepto	**Super-Concepto**	**Descripción**	**Propósito**
Artefacto	Elemento	Parte diferenciada de un producto software que es creada o modificada por las actividades. Puede ser un documento (textual o gráfico), un componente COTS, o un módulo de código. Ejemplos: documento de especificación de requisitos, plan de calidad, módulo de clase, rutina, informe de pruebas, manual de usuario. Sinónimos: elemento software, producto de trabajo, item de producto.	Definir la estructura interna y composición del software.
Producto	Concepto	Aplicación software que está siendo mantenida. Es un conglomerado de diversos artefactos. Sinónimo: software.	Mantenerlo.
Versión	Concepto	Un cambio en la línea base de un producto. Puede ser una versión completa, *upgrade*, *release* o actualización, o un simple parche en el código.	Implantar el proceso de gestión de configuración

Tabla 1. Subontología de los productos: Glosario de conceptos

Nombre	**Descripción**	**Cardi-nalidad**	**Tipo**	**Obligatorio**	**Peso**
Madurez (artefacto)	Etapa del ciclo de vida del producto	1	MaturityType	Si	1
Tamaño (artefacto)	Medida cualitativa del tamaño.	1	SizeMeasure	Si	1
Composición (artefacto)	Nivel de abstracción de los artefactos que lo forman.	1	CompositionType	Si	1
Tipo de aplicación (artefacto)	Tipo de aplicación. Por ejemplo: de gestión, científica, de control, etc.	1	ApplicationType	Si	1
Calidad (artefacto)	Medida cualitativa de la calidad, especialmente de la documentación.	1	MeasureQ	Si	1
Edad (artefacto)	Número de años desde que se obtuvo la primera versión.	1	Integer	Si	1
Componente (I/F)	Artefactos en los que se divide el producto.	0..*	Has-decomposition [artefacto]	No	1

Tabla 2. Subontología de productos: Tabla de atributos

Para cada producto, al cual una empresa de mantenimiento del software debe darle mantenimiento, se rellenará una tabla que describa las características del producto software según los atributos especificados en la Tabla 2. Esta información podrá utilizarse para comparar la similitud entre distintos productos tal y como se explica en la siguiente sección.

3. Reutilización del conocimiento

La información de un sistema de gestión del conocimiento debe ser reutilizada, en caso contrario no se obtendría todo el beneficio que se espera de dicho sistema. Para potenciar la reutilización de la información almacenada en nuestro sistema utilizamos Razonamiento Basado en Casos (CBR), técnica ampliamente utilizada con el fin de encontrar la mejor solución a un problema entre un grupo posible de soluciones [Niknafs et al. 03]. Por otro lado, el hecho de que REFSENO utilice funciones de similitud ha facilitado el uso de CBR en nuestra aplicación.

A continuación se describe con un ejemplo cómo el sistema utiliza las funciones de similitud. En este caso se pretende comparar dos instancias de productos: *i* y *q*.

El primer paso es calcular las funciones de similitud para cada capa. En el caso del Producto, como se puede apreciar en la Tabla 2, sólo hay dos capas: artefacto e I/F:

$$Sim(product)(i,q) = W_{artif} * \mathbf{sim_{artif}(product)\ (i, q)} + W_{I/F} * \mathbf{sim_{I/F}(product)\ (i, q)}$$

Las funciones de similitud locales se calculan mediante la suma de las funciones de similitud de cada tipo de atributo perteneciente a esta capa. Finalmente, cada función de similitud local es normalizada resultando un valor en el rango [0,1]. Por ejemplo, en el caso de la capa artefacto del concepto producto es necesario conocer las funciones de similitud de los tipos: *MaturityType*, *SizeMeasure*, *CompositionType*, *AppplicationType*, *MeasureQ* e *Integer* para poder calcular la suma de todas ellas.

REFSENO proporciona varios tipos de datos predefinidos y sus respectivas funciones de similitud. Por ejemplo, el tipo "Integer" tiene la siguiente función de similitud para comparar las instancias *i* y *q*:

$$Sim(i,q) = 1 - \frac{|i-q|}{(\max value - \min value)}$$

donde *minvalue* y *maxvalue* representan el menor y mayor valor correspondiente al rango de valores.

Los tipos de similitud para aquellos tipos definidos por el usuario, tales como "MaturityType", también deben ser descritos. Por ejemplo, "MaturityType" es una taxonomía formada por 4 etiquetas: inicial, evolución, servicio y retirada y su función de similitud es la siguiente:

Sim(i,q):	1	si i=q
	0.5	si i = inicial & q = evolución o viceversa
	0.25	si i = inicial & q = servicio o viceversa
	0	si i = inicial & q = retirada o viceversa
	0.5	si i = evolución & q = servicio o viceversa
	0	si i = evolución & q= retirada o viceversa

Después de calcular las funciones de similitud locales $\mathbf{sim_{artif}(product)(i, q)}$ y $\mathbf{sim_{I/F}(product)\ (i, q)}$ se deben calcular las funciones de similitud globales asignando valores a W_{artif}, and $W_{I/F}$, dependiendo de las necesidades del usuario. Por ejemplo si el sistema quiere comparar la similitud entre dos productos de acuerdo a sus propias características, el valor de W_{artif} deberá ser maximizado y $W_{I/F}$ minimizado ya que la suma de sus pesos debe ser siempre 1. De esta forma el sistema adapta los pesos según las prioridades de los usuarios.

El sistema usa principalmente las funciones de similitud para comparar productos software, y peticiones de mantenimiento. El objetivo principal de comparar productos es predecir futuras demandas de mantenimiento, ya que productos con características similares demandan muy a menudo el mismo tipo de modificaciones. Tal y como explican en [Bennett00], si los cambios pueden ser anticipados estos podrán realizarse de una forma más planificada y de esta manera su costo y esfuerzo disminuirán.

La finalidad de comparar las peticiones de mantenimiento es la reutilización de soluciones para problemas similares y así evitar repetir los mismos errores.

4. Conclusiones y trabajo futuro

El proceso de mantenimiento de software genera gran cantidad de conocimiento que debe ser procesado y gestionado con el fin de disminuir costes y esfuerzo. Sin embargo, antes de gestionarlo los diferentes tipos de información y sus relaciones deben ser especificados. Para ello hemos desarrollado una ontología que define los conceptos vinculados al mantenimiento del software y sus relaciones. Un factor a destacar de la ontología es su aspecto práctico, ya que ha sido utilizada en varios proyectos que nos han permitido detectar algunas de sus limitaciones y depurarla progresivamente. Por ejemplo, la ontología de flujos de trabajo surgió a raíz de analizar los aspectos dinámicos de un proyecto de mantenimiento.

Otra aportación importante de este artículo es la explicación de cómo llevar a cabo la implementación

de la ontología puesto que aunque actualmente están surgiendo cada vez más trabajos que describen diseños ontológicos (ver www.semanticweb.org) todavía pocos explican cómo realizar la implementación de la misma en un sistema. Por último, se ha ilustrado cómo se usan las funciones de similitud para potenciar la reutilización de la información y concretamente de casos similares que puedan ayudar a los ingenieros del mantenimiento a aprovechar las lecciones aprendidas que la compañía posee. De esta forma se evita que se busquen soluciones a problemas que ya han sido resueltos y se pretende que el trabajo del ingeniero del conocimiento sea más cómodo y eficiente.

Con el fin de mejorar dicho sistema se hizo una encuesta a varios encargados del mantenimiento en la cual se les consultaba sobre qué funcionalidades añadirían. Casi todos los encargados contestaron que les gustaría que la aplicación tuviera un mecanismo que les indicara dónde encontrar la información necesaria para realizar una tarea. Actualmente se está trabajando en este objetivo y una vez terminada, la herramienta será nuevamente evaluada por los ingenieros del mantenimiento.

Agradecimientos

Este trabajo es financiado por el proyecto MAS (TIC2003-02737-C02-02, Ministerio de Ciencia y Tecnología de España), el proyecto ENIGMAS (Junta de Comunidades de Castilla-La Mancha, Consejería de Educación y Ciencia, referencia PBI-05-058) y el proyecto MECENAS (PBI06-0024), Junta de Comunidades de Castilla-La Mancha, Consejería de Educación y Ciencia (España).

Referencias

[Banker et al. 91] Banker, R.D, Datar, S.M., y Kemerer, C.F. A Model to Evaluate Variables Impacting the Productivity of Software Maintenance Project. *Management Science*. Vol. 37, No. 1, pp: 1-18. (1991).

[Banker et al. 02] Banker, R. D., Datar, S.M., Kemerer, C.F. and Zweig D. Software Errors and Software Maintenance Management. *Kluwer Academic Publishers*. (2002).

[Bennet00] Bennett, K. H., Rajlich, V.T. Software Maintenance and Evolution: a Roadmap. *The Future of Software Engineering, Proceedings ICSE*. Limerick, Ireland. pp: 75-87, (2000).

[Card90] Card, D. N., Glass, R.L. Measuring Software Design Quality. Englewood Cliffs, USA, *Prentice Hall*, (1990).

[de Loff97] De Looff, L., De Looff, L. A., Information Systems Outsourcing Decision Making: a Managerial Approach. Hersey, Idea Group Publishing, (1997).

[Deridder02] Deridder, D. A Concept-Oriented Approach to Support Software Maintenance and Reuse activities. *Proceedings of the 5th Joint Conference on Knowledge Based Software Engineering* (JCKBSE'02), IOS Press. 2002.

[Falbo et al. 99] Falbo, R. A., Menezes, C.S., Rocha, A.R. Using Knowledge Serves to Promote Knowledge Integration in Software Engineering Environments. *Proceedings 11th International Conference on Software Engineering and Knowledge Engineering* (SEKE' 99). pp: 170-175, (1999).

[Gómez-Pérez et al. 04] Gómez-Pérez, A., M. Fernández-López, Corcho, O. Ontological Engineering. ISBN 1-85233-551, Springer-Verlag London, (2004).

[Gruber93] Gruber, T. R. (1993). A Translation Approach to Portable Ontology Specification. *Knowledge Acquisition*. Vol. 5, No. 2, pp: 199-220, (1993).

[Hikita&Matsumoto01] Hikita, T., Matsumoto, M.J., Business Process Modelling Based on the Ontology and First-Order Logic. *Proceedings of the Third International Conference on Enterprise Information Systems* (ICEIS). pp: 717-723, (2001).

[Kitchenham et al. 99] Kitchenham, B. A., Travassos, G. H., Mayrhauser, A., Niessink, F., Schneidewind, N. F., Singer, J., Towards an Ontology of Software Maintenance. *Journal of Software Maintenance: Research and Practice*. Vol. 11, pp: 365-389, (1999).

[Meadche et al. 03] Maedche, A., Motik, B., Stojanovic, L., Studer, R., Volz, R., Ontologies for Enterprise Knowledge Management. *IEEE Intelligent Systems*, pp: 26-33, (2003).

[Niknafs et al. 03] Niknafs, A., Shiri, M., Javidi, M. A Case-Based Reasoning Approach in E-Tourism: Tour Itinerary Planning. *Proceedings of 4th International Workshop on Theory and Applications of Knowledge Management* (DEXA'03). Prague, Czech Republic. (2003).

[Nonaka&Takeuchi95] Nonaka, I., Takeuchi, H. The Knowledge Creation Company: How Japanese Companies Create the Dynamics of Innovation. *Oxford University Press*. 1995.

[Oliveira et al. 03] Oliveira, K. M., Anquetil, N., Dias, M.G., Ramal, M., Meneses, R. Knowledge for Software Maintenance. *Fifteenth International Conference on Software Engineering and Knowledge Engineering* (SEKE), pp: 61-68, (2003).

[Pigoski97] Pigoski, T., Practical Software Maintenance. *John Wiley & Sons*, (1997).

[Polo et al. 99] Polo, M., Piattini, M., Ruíz, F. and Calero, C. MANTEMA: a Complete Rigorous Methodology for Supporting Maintenance Based on The ISO/IEC 12207 Standard. *Proceedings of the Third European Conference on Software Maintenance and Reengineering*. Amsterdam, (1999).

[Ruiz et al. 02] Ruiz, F., García, M., Piattini, M., Polo, M. Environment for Managing Software Maintenance Projects. USA, *Idea Group Publishing*, (2002).

[Ruiz et al. 04] Ruiz, F., Vizcaíno, A., Piattini, M., García, F. An Ontology for the Management of Software Maintenance Projects. *International Journal on Software Engineering and Knowledge Engineering*. Vol. 14. No. 3. pp: 323-346, (2004).

[Rus&Lindvall02] Rus, I., Lindvall, M. Knowledge Management in Software Engineering. *IEEE Software*. Vol. 19, pp: 26-38, (2002).

[Souza et al. 04] Sousa, K. D., Anquetil, N., Oliveira, K.M. Learning Software Maintenance Organizations. *Proceedings Advances in Learning Software Organizations* (LSO) LNCS 3096. pp: 20-21, (2004).

[Tautz98] Tautz, C. G., Von Wangenheim. REFSENO: A Representation Formalism for Software Engineering Ontologies. *Fraunhofer IESE-Report*, (1998).