Apoyo a la Decisión en Ingeniería del Software

J. Dolado, J.J. Cuadrado y D. Rodríguez García

Presentación

Nos encontramos ante una nueva edición del taller "Apoyo a la decisión en ingeniería del software". A través de todos estos años se han ido presentando nuevos temas de investigación. En esta ocasión también hemos conseguido que varios colaboradores expongan las investigaciones que tienen en curso. Los temas son variados: coordinación de proyectos, visualización de la gestión, ontologías, gestión de equipos, XBRL, estimación y SWEBOK. Los editores agradecen el esfuerzo realizado por los participantes, así como las entidades que han financiado a los investigadores: el Ministerio de Educación y Ciencia bajo el proyecto TIN2004-06689-C03-01 y la Comunidad Europea bajo el proyecto MIF-CT-2006-039212 (Alain Abran).

Zaragoza a 11 de septiembre de 2007

La aplicación de metáforas paisajísticas en el control del proceso de desarrollo software

Amaia Aguirregoitia
Depto. de Lenguajes y Sistemas Informáticos
Universidad del País Vasco,Escuela Universitaria de Ingeniería Técnca Industrial de Bilbao
La Casilla ,3 48012 Bilbao (Vizcaya)
amaia.aguirregoitia@ehu.es
Javier Dolado
Depto. de Lenguajes y Sistemas Informáticos
Universidad del País Vasco, Facultad de Informática
20.009 Donostia - San Sebastián, Spain
dolado@si.ehu.es

Resumen

En este artículo se presenta un modelo de representación gráfica para el control y la gestión del proceso de desarrollo de software basado en las áreas de gestión de proyectos, representación visual de datos y aplicación de metáforas. Partiendo de un conocido modelo de gestión clásico para la definición de actividades (planificación, organización, gestión de personal, dirección y control) y considerando como magnitudes básicas a controlar el proceso, el producto y los recursos, se han definido los indicadores básicos para efectuar el control del proceso de desarrollo en cada una de éstas magnitudes. Posteriormente, se ha buscado una representación gráfica que represente de un modo completo, fácil de comprender y lo más natural posible, la evolución del proceso de desarrollo mediante la visualización de los valores de estos indicadores, para lo cual ha sido necesario trabajar tanto en el campo de la visualización como en el de la representación metafórica.

Se trata de un trabajo que se enfrenta a los complejos problemas del control del proceso de desarrollo software tratando de obtener una visión sencilla de la situación que nos permita valorar y comparar resultados de la gestión del proceso a través de una única pantalla que presente de forma integrada toda la información requerida. En las posteriores fases del trabajo podrían abordarse otros posibles usos de las metáforas, así como modificaciones en aspectos de la representación como el color y otros atributos e incluir funcionalidades que permitan la interacción del usuario para opciones de filtrado, vuelo aéreo o zoom.

La representación actual si bien es sencilla en su forma, permite la detección de problemas dispares y de diversa complejidad como son el seguimiento del plan, el cumplimiento de hitos, la evaluación de los procedimientos tanto de control de errores como de aprobación y ejecución de modificaciones, la evaluación de la calidad del producto o la detección de recursos infrautilizados.

Palabras clave : Gestión de proyectos, visualización del software, Gestión de la calidad del Software.

1. Problemas de la gestión del software

La gestión de un proyecto software podemos definirla como el conjunto de procedimientos, prácticas, tecnologías y conocimientos que tienen como objetivo la planificación, organización, gestión de personal, dirección y el control requeridos para gestionar adecuadamente un proyecto de ingeniería del software. [13]La gestión de un proyecto software se realiza considerando como magnitud elemental el tiempo y analizando en un momento específico cuál es la situación actual del proyecto teniendo en cuenta tanto los recursos como el conjunto de tareas implicadas en el proceso de elaboración de un producto software. El problema es abordado con una triple visión de las entidades producto-

proceso-organización de recursos [4] que evolucionan en el tiempo y utilizando una serie de variables o mediciones a controlar. En la definición de estas variables aparecen una serie de problemas relacionados con la medición del software que se resumen del siguiente modo:

- En la gestión del Software se ven involucradas diversas áreas de conocimiento, como son las medidas del software, métodos de validación de medidas, modelos de estimación de recursos y diversas filosofías de desarrollo.
- En el proceso de desarrollo existe una gran diversidad de metodologías, herramientas, tecnologías que evolucionan rápidamente y éste se ve afectado por factores como el lenguaje de programación o el entorno utilizado.
- Las medidas del software a representar requieren de análisis causales que expliquen los eventos cuantificados.
- Las métricas deberían ser fáciles de calcular y automatizar y obtenerse en las primeras fases del proceso para mayor validez.
- Las métricas se deben contemplar conjuntamente con otros atributos del producto como por ejemplo, la calidad del producto software.
- Para canalizar la energía de la organización y enfocarla en las áreas estratégicas con mayor potencial de retorno es importante encontrar un número pequeño de métricas simples, lo cual conlleva una gran dificultad.
- Para analizar el proceso de desarrollo se deben considerar tanto atributos internos como externos (en función de cómo se relaciona el proceso, producto o recurso con su entorno).
- Validar una métrica necesita de amplios datos empíricos.
- Una métrica debe tener un modelo de soporte, ser útil, representativa y tener un objetivo concreto.[5]
- El proceso y el producto conllevan diferentes fases y todas ellas deben considerarse.

En cuanto a los problemas de gestión específicos del software que nos conciernen, se han considerado como los más relevantes:

- Los resultados obtenidos de los esfuerzos de gestión son difíciles de cuantificar.
- Generalmente existe una urgencia en las entregas y el cumplimiento de hitos que puede llevar a descuidar el proceso y dar prioridad al producto entregable llegando al extremo de descuidar la calidad.
- El seguimiento de las modificaciones en los requerimientos y la trazabilidad de los mismos es en muchas ocasiones descuidado.
- Para llevar a cabo la institucionalización de los procesos de mejora continua del software se requiere de la mentalización de todos los implicados.
- La gestión de un área del proceso (Gestión de riesgos) se debe realizar considerando las áreas de nivel inferior que son requisito para que la primera sea correcta (por ejemplo, Gestión de requisitos).
- La mayoría de las actividades son realizadas por hombres, lo que requiere flexibilidad y cooperación, y se requiere de un alto grado de consideración hacia el factor humano.

2. Problemas de la visualización del software

La mayoría de los trabajos de visualización en el campo del Software han estado enfocados a la representación gráfica de la arquitectura, diagramas de ejecución en tiempo real con llamadas entre funciones o módulos y otras relaciones en modo grafo, o representación en modo de estadísticas y diagramas más o menos tradicionales sobre ciertos aspectos del código [1] o rendimiento enfocados a la mejora del código[14]. Existen sistemas, como Imsovision[8]

que han aplicado técnicas de realidad virtual que utilizan metáforas visuales con el objetivo de entender mejor el programa y facilitar su mantenimiento y desarrollo y que constatan la dificultad de representar los objetos, atributos, métricas y relaciones entre objetos especialmente con metáforas.[9]

Los problemas encontrados en este campo van desde la representación semántica, o de los modelos cognitivos necesarios para la representación e interpretación, la necesidad de representar la evolución del producto, así como múltiples vistas de los mismos datos, la necesidad de visualizaciones tanto estáticas como dinámicas o la gran cantidad de información a manejar.[7][10]

En nuestro caso, y a diferencia de los mencionados trabajos dentro del campo de la visualización del software el reto es la representación gráfica en sí. Se quiere encontrar una representación gráfica que muestre los datos de modo simple y potente y que sirva, gracias al diseño elegido, para describir, relacionar y razonar, dando soporte a la gestión del proyecto.

En el área de visualización orientado a la gestión de proyectos software, se han desarrollado proyectos que utilizan desde los típicos como ruedas del tiempo en 2D y 3D [2] o Glyphs especialmente diseñados (objetos gráficos o símbolos que representan datos o información a través de parámetros visuales que son espaciales, retinales o temporales).

3. El proyecto

3.1. La Espacialización y la metáfora

En el proyecto utilizaremos una clase de visualización de la información conocida como “spatializations” [11], o espacialización , en la cual información no espacial , en este caso del proyecto software, es representada como una entidad extendida en el espacio, en nuestro caso , como una banda temporal en 2D.

Pettifer y West [12] en su proyecto identifican los beneficios de de las metáforas con el mundo real ya que hacen uso de las habilidades espaciales y de percepción adquiridas y usadas inconscientemente añadiendo familiaridad y realismo al sistema.

Existen trabajos en el área de visualización de software como el realizado por Young[15] que utiliza una visualización abstracta en 3D para representar módulos y el sistema de llamadas. Nuestro proyecto utiliza la visualización utilizando la metáfora con el mundo real, más en línea con trabajos realizado por Knight y Munro [6] que han creado una escena más familiar al ojo humano representado el sistema como una ciudad donde por ejemplo, los edificios representan los métodos.

En el proyecto se utiliza para la metáfora visual un paisaje. Sobre una banda temporal se representan los eventos, cambios esenciales o alteraciones producidos en los principales aspectos del proceso software buscando la metáfora mediante la asociación de estos aspectos bien con algún objeto, o bien algún atributo a propiedad de algún objeto del paisaje.

3.2. Los indicadores seleccionados

El siguiente paso ha sido determinar los eventos, atributos y propiedades a representar, para lo cual, inicialmente se ha considerado inicialmente un número pequeño de valores con el objetivo de:

- Realizar el seguimiento de la ejecución del plan establecido y a controlar la situación del proyecto en un momento dado así como sus posibles desviaciones. (proceso)
- Evaluar la calidad del desarrollo efectuado. (producto)
- Identificar situaciones excepcionales o de especial riesgo. (proceso-producto)
- Evaluar la utilización de los recursos y de la productividad y eficiencia. (recursos)

La Tabla 1- Lista de indicadores clasificados por entidades resume los indicadores seleccionados. En un primer estudio sobre casos reales se ha puesto de manifiesto que en la mayoría de los casos, los indicadores seleccionados son relevantes y o bien se dispone del dato en la actualidad o bien sería posible su incorporación.

MEDIDA	PRODUCTO					PROCESO				RECURSOS		
	Errores ,Fallos	TMF	Crecimiento ,Fiabilidad	Fallos restantes	Completud	Complejidad	Control de la gestión	Cobertura	Eval. Riesgos, beneficios y costes	Personal	Hardware	Software
Horas persona dedicadas reales							X		X	X		
Horas persona dedicadas planificadas							X		X	X		
Desviación en Euros respecto al ppto.							X		X	X	X	X
tº total de corrección errores REQ	X						X		X	X		
Horas persona de corrección errores REQ	X						X		X	X		
tº total de corrección errores DIS	X						X		X	X		
Horas persona de corrección errores DIS	X						X		X	X		
tº total de corrección errores COD	X						X		X	X		
Horas persona de corrección errores COD	X						X		X	X		
tº total de corrección errores DOC	X						X		X	X		
Horas persona de corrección errores DOC	X						X		X	X		
Horas persona no productivas							X		X	X		
Hito incumplido							X					
Horas desviación por hito							X		X			
Revisión efectuada REQ							X	X				
Revisión efectuada DIS							X	X				
Revisión efectuada COD							X	X				
Revisión efectuada DOC							X	X				
Gestión de riesgos realizada									X	X		
Horas/persona para gestión de riesgos							X		X	X		
Horas/pers. reutilizadas(proy. anteriores)									X	X		
Nº modificaciones solicitadas COD							X					
Nº modificaciones implementadas COD			X		X	X				X		
Nº modificaciones rechazadas COD							X					
Nº modificaciones solicitadas DIS							X					
Nº modificaciones implementadas DIS			X		X	X				X		
Nº modificaciones rechazadas DIS							X					
Nº modificaciones solicitadas REQ							X					
Nº modificaciones implementadas REQ			X		X	X				X		
Nº modificaciones rechazadas REQ							X					
Horas persona dedicadas a modif COD						X			X	X		
Horas persona dedicadas a modif DIS						X			X	X		
Horas persona dedicadas a modif REQ						X			X	X		
Nº errores detectados antes entrega	X		X	X			X					
Nº errores detectados después entrega	X		X	X			X					
% errores detectados después entrega	X		X	X								
tº medio resolución errores						X	X					

Tabla 1- Lista de indicadores clasificados por entidades

3.3. La representación integral

Para la representación de los indicadores seleccionados se ha buscado una metáfora con el mundo real tratando de relacionar la métrica representada con un objeto o atributo que se le asemeje en mayor o menor grado en alguno de sus aspectos. A continuación se comentan algunos de las metáforas utilizadas.

En el proceso de desarrollo se realiza un esfuerzo que medido en horas y acumulado a lo largo del tiempo guarda similitud en su forma con una montaña cuya altitud se irá incrementando para representar las horas acumuladas en cada momento del tiempo. De manera cuantitativa podemos visualizar en el eje vertical las horas y de manera cualitativa podemos observar la pendiente y las variaciones en el esfuerzo tanto estimado como real. Los "agujeros" de asignación de recursos serán los surcos en el camino cuya profundidad nos expresa las horas sin asignar.

Otro ejemplo sería el de los hitos. Un hito es un evento significante en el proyecto, generalmente la consumación de un entregable que suele ir asociado a una fecha objetivo. Es necesario algo significante, claramente visible y que permita representar en el mismo objeto la desviación con respecto a la planificación inicial. Se utilizarán las estrellas que representan hitos con desviaciones, que dejarán tras de sí una estela cuyo tamaño representa la desviación en tiempo con respecto a lo inicialmente previsto. Esta metáfora permite representar desviaciones tanto positivas como negativas según sea el sentido de la estela.

Los inevitables cambios en el producto implicarán ajustes en el tiempo y para representarlo se utilizará como metáfora una nube que representará con su tamaño el tiempo dedicado a modificaciones. Todas las modificaciones no serán igualmente peligrosas, por lo que hay nubes sin lluvia (cambios de código), con ella (diseño) y con rayos (requerimientos).

Otro aspecto importante es la determinación y evaluación de aquellos aspectos que pueden afectar negativamente al proyecto. Lo crítico es determinar en qué momento y cuánto esfuerzo se dedica a la gestión de riesgos y la metáfora propuesta son las piedras cuya ubicación y tamaño nos dará la información necesaria.

En el área de resolución de errores se quiere representar el tiempo medio de resolución de éstos junto con el número de errores detectados por el equipo de desarrollo en número y en porcentaje sobre el total. Se representa como un sol (entendiendo que la resolución de errores trae "luz" al proyecto) cuyo desplazamiento respecto al origen será el tiempo medio de resolución y el resto de datos aparecen en su interior.

En el Gráfico 1- Representación integral aparece la representación elegida en el proyecto y en la Tabla 2- Indicadores y su representación se indica la relación entre todos los indicadores representados y su representación metafórica.

3.4. Las preguntas a las que responde el proyecto

Se trata de conocer si estamos en una situación de riesgo, si nuestro proceso de desarrollo está siendo eficiente o en qué aspectos podemos mejorarlo. La representación responde efectivamente a las siguientes preguntas:

Acerca del proceso:

- ¿Cuál es la desviación actual del proyecto en horas con respecto a lo establecido?
- ¿Cuál ha sido la evolución de esta desviación en horas?
- ¿Cuál es la desviación económica actual del proyecto con respecto a lo establecido?
- ¿Cuál ha sido la evolución de esta desviación económica?
- ¿En qué momento ha tenido lugar un incumplimiento de hito?
- ¿Cuándo estaba previsto el cumplimiento de este hito incumplido?
- ¿Qué tipos de errores está detectando nuestro sistema de control de errores?
- ¿Cuánto tiempo y recursos estamos dedicando a la corrección de cada tipo de errores?
- ¿Cuántas modificaciones se están solicitando, implementando y rechazando?

- ¿Cuál es el esfuerzo dedicado a la gestión de riesgos en horas y en qué momentos?
- ¿Cuál es el grado estimado de trabajo reutilizado en horas/persona?
- ¿Existen recursos infrautilizados medido en horas/persona?

Acerca del producto:

- ¿Cuál es la calidad del producto entregada, medida en número de errores totales, detectados por el usuario?
- ¿Cuántas modificaciones de cada tipo ha sufrido en producto y cuánto tiempo se ha empleado en las mismas?
- ¿Cuántas revisiones y en qué momentos se han efectuado a ese producto?
- ¿Cuántas modificaciones pendientes tiene el producto?

Acerca de los recursos

- ¿Cuál es el grado de eficiencia de los recursos utilizados, existen infrautilizaciones?
- ¿En qué momento se han producido los tiempos muertos y cuánto tiempo han durado?

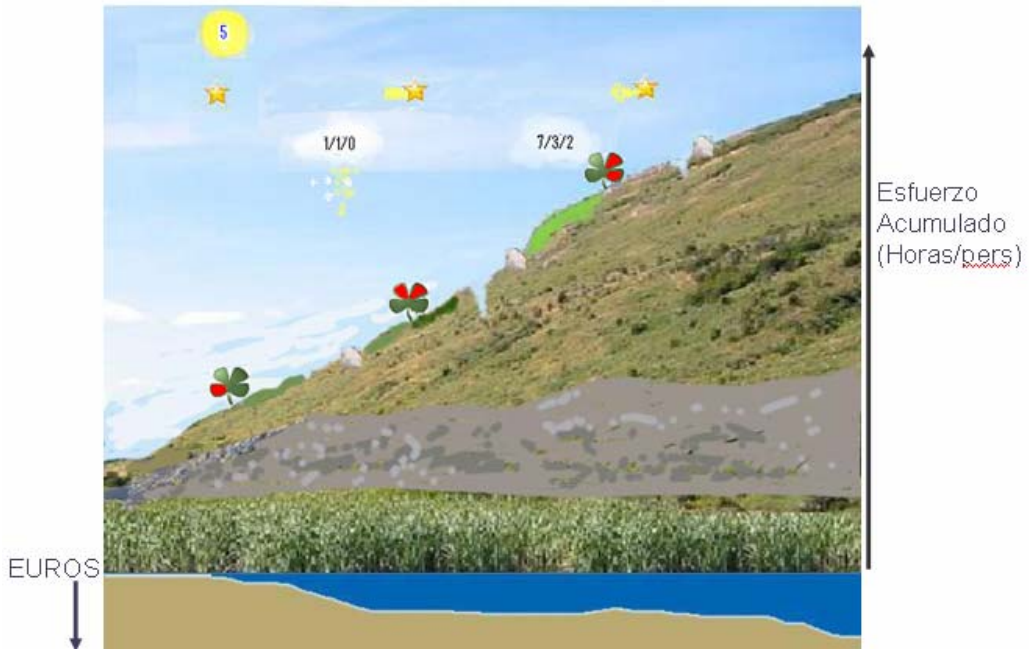

Gráfico 1- Representación integral

1	Horas persona dedicadas reales	Altitud del monte representado en color marrón
2	Horas persona dedicadas planificadas	Altitud del monte indicada en gris
3	Desviación en Euros respecto al ppto.	Profundidad del pozo en azul.
4	Tiempo de corrección errores de requerimientos	Anchura del musgo verde más claro.
5	Horas/persona corrección errores de requerimientos	Altura del musgo verde más claro.
6	Tiempo de corrección errores de diseño	Anchura del musgo verde claro.
7	Horas/persona de corrección errores de diseño	Altura del musgo verde claro.
8	Tiempo de corrección errores de codificación	Anchura del musgo verde oscuro.
9	Horas persona de corrección errores de codificación	Altura del musgo verde oscuro.
10	Tiempo de corrección errores de documentación	Anchura del musgo verde más oscuro.

11	Horas/persona corrección errores de documentación	Altura del musgo verde más oscuro.
12	Horas persona no productivas	Altura de surco en el monte
13	Hito incumplido	Estrella cuya posición destaca la existencia de un hito incumplido.
14	Desviación en horas de cumplimiento por hito	Estela de la estrella que indica la existencia del hito.
15	Revisión de requerimientos efectuada	Trébol con la primera de sus hojas por la izquierda en rojo
16	Revisión de diseño efectuada	Trébol con la segunda de sus hojas por la izquierda en rojo.
17	Revisión de codificación efectuada	Trébol con la tercera de sus hojas por la izquierda en rojo.
18	Revisión de documentación efectuada	Trébol con la cuarta de sus hojas por la izquierda en rojo.
19	Existencia de tarea realizada de gestión de riesgos.	Piedra representada en el monte.
20	Horas persona dedicadas a gestión de riesgos	Anchura de la piedra representada en el monte.
21	Horas persona reutilizadas (proyectos anteriores)	Altura sobre la cual se empieza a dibujar el monte.
22	Nº modificaciones de código solicitadas.	Nube sin lluvia, primer dígito indicado.
23	Nº modificaciones de código implementadas.	Nube sin lluvia, segundo dígito indicado.
24	Nº modificaciones de código rechazadas	Nube sin lluvia, tercer dígito indicado.
25	Nº modificaciones de diseño solicitadas.	Nube con lluvia, primer dígito indicado.
26	Nº modificaciones de diseño implementadas.	Nube con lluvia, segundo dígito indicado.
27	Nº modificaciones de diseño rechazadas	Nube con lluvia, tercer dígito indicado.
28	Nº modificaciones de requerimientos solicitadas	Nube con lluvia y rayo, primer dígito indicado.
29	Nº modificaciones de requerimientos implementadas	Nube con lluvia y rayo, segundo dígito indicado
30	Nº modificaciones de requerimientos rechazadas	Nube con lluvia y rayo, tercer dígito indicado
31	Horas persona dedicadas a modificaciones de código	Altura de la nube sin lluvia.
32	Horas persona dedicadas a modificaciones de diseño.	Altura de la nube con lluvia.
33	Horas/pers. dedicadas a modificaciones de requisitos.	Altura de la nube con lluvia y rayo.
34	Nº errores detectados antes entrega.	(Requeridos para calculo y no visualizado)
35	Nº errores detectados después entrega.	Segundo dato representado dentro del sol.
36	% errores detectados después entrega.	Primer dato representado dentro del sol.
37	Tiempo medio resolución errores.	Desplazamiento del sol respecto al punto origen

Tabla 2- Indicadores y su representación

4. Líneas futuras

Durante el desarrollo de este modelo hemos tenido que desestimar una serie de indicadores relevantes en beneficio de una representación simple debido a la gran cantidad de información ya representada. Existen datos especialmente relevantes para poder evaluar el proceso de desarrollo acerca de aspectos como el número de errores (importante para estudiar el patrón de los mismos), recursos infrautilizados medidos en número de personas y no sólo en horas, esfuerzo dedicado a las revisiones en horas (actualmente sólo se visualiza en qué momento para saber que han sido efectuadas sin valorar los recursos utilizados para ello), grado de cobertura de las pruebas (actualmente se visualiza en esfuerzo, pero no la cobertura de las mismas), tiempos medios entre fallos (para conocer la fiabilidad del producto), horas hombre por fallo detectado, aportación de nuestro producto a la reutilización y todos éstos sin entrar a valorar la posibilidad de incluir métricas de complejidad del software.

Indudablemente la inclusión de esta información enriquecería la visión, aunque es probable que la inclusión de ésta sea más apropiada a un segundo nivel o bien a través de funcionalidades implementadas en el software como navegación, zoom o incluso vuelo estilo Googleearth con datos de proyecto.

Para la implementación se ha optado por el desarrollo de la aplicación en el entorno gráfico de Java utilizando las librerías awt y swing así como API's para desarrollo gráfico que ofrecen la flexibilidad requerida para esta representación. Los siguientes pasos podrían ser decidir las funcionalidades de navegación a implementar a partir de esta representación integral para añadir funcionalidades de análisis y exploración así como el nivel de detalle de los datos que serán ofrecidos en este segundo nivel. Este trabajo se complementará con otras representaciones que ofrecerán la visión por actividades, tareas o producto y eliminando como base el eje temporal.

Agradecimientos

Este trabajo se ha desarrollado bajo el proyecto TIN2004-06689-C03-01.

Referencias

[1] Baker Marla J. ;Eick ,Stephen G. . “Space filling SW visualization”. Journal of visual languages and computing 1995,6,119-133.

[2] Chuah Mei C., Stephen G. Eick. “ Information Rich glyphs for Software management data”. (1998)

[3] Cugola Gianpaolo; Ghezzi,Carlo. “Software Processes:a Retrospective and a Path to the Future”.International Conference on Software Process (Lisle, IL, 14-17 June 1998).

[4] Fenton, Norman E.; Neil, Martin . Software metrics Roadmap , (2000).

[5] Fuggetta Alfonso. Software Process: A Roadmap . “The Future of Software Engineering” Finkelstein Anthony (Ed.). ACM Press 2000

[6] Knight, C.; Munro, M. “Comprehension with[in] Virtual environment visualizations.” Proceedings in the IEEE 7th International Workshop on program comprehension, Pittsburgh, Pa May 5-7,1999.

[7] Koschke,Rainer. “Software visualization for reverse engineering” Artículo del libro Software visualization 138-150 Ed. Springer(2002)

[8] Maletic ,Jonathan I.;Leigh, Jason; Marcus ,Andrian;Dunlap Greg. “Visualizing Object-oriented software in Virtual reality”. Proceedings of the 9th IEEE International Workshop on Program Comprehension (IWPC2001), Toronto, Canada, May 12-13, 2001, pp. 26-35.

[9] Maletic, Jonathan I., Marcus, Andrian;Feng Louis .”Source Viewer 3D(sv3D)-A Framework for Software Visualization” International Conference on Software Engineering .Proceedings of the 25th International Conference on Software Engineering.Portland, Oregon .Pages: 812 – 813.ISBN - ISSN:0270-5257 , 0-7695-1877-X (2003)

[10] Maletic, Jonathan I., Marcus, Andrian;Collard Michael L. “A task oriented View of Software Visualization”. VISSOFT Proceedings of the 1st International Workshop on Visualizing Software for Understanding ISBN:0-7695-1662-9 p. 32(2002)

[11] Montello, Daniel R. ; Fabrikant, Sara Irina; Ruocco, Marco; and Richard S. Middleton. “Testing the First Law of Cognitive Geography on Point-Display Spatializations”. Spatial Information Theory Foundations of Geographic Information Science; International Conference. COSIT 2003, LNCS 2825, pp. 316–331, (2003)

[12] Pettifer ,S.R. ;West, A.J. . “Deva: A coherent operating environment for large scale VR applications”. Myron Kreuger, Grigore Burdea, Henry Fuchs, and Michael Zyda, editors, *Proc. Virtual Reality Universe ‘97*, April 1997. (1997)

[13] Thayer Richard H.. Software engineering Project management . IEEE Computer society (1997).

[14] Umphressa, David A.; Hendrix T. Dean, Cross II James H. , Maghsoodloob Saeed.”Software visualizations for improving and measuring the comprehensibility of source code”. Science of Computer Programming 60 (2006) 121–133

[15] Young P.;Munro M. “Visualizing Software in Virtual reality”.Proceedings on the IEEE3 International Workshop on program comprehension, Ischia, Italy Juen 24-26 1998, pp19-26 (1998)

Medición práctica de la coordinación utilizando GQ(I)M y CMMi

Mª Concepción Presedo García
Dept. de Lenguajes y Sistemas Informáticos
EUITI Bilbao
UPV/EHU
48012 Bilbao
conchi.presedo@ehu.es

J. Javier Dolado Cosín
Dept. de Lenguajes y Sistemas Informáticos
Facultad de Informática
UPV/EHU
20009 Donostia- San Sebastián
javier.dolado@ehu.es

Resumen

En el presente artículo se realiza una propuesta para la medición práctica de la coordinación dentro del proceso software utilizando como guías las prácticas clave de los modelos de madurez CMM, el proceso de medición Goal-Questions-Indicators-Measures, más conocido como GQ(I)M, así como una aplicación software que implementa el proceso de medición PSM.

1. Propósito y estructura

En esta primera sección describimos la motivación y estructura del artículo.

En cuanto a la motivación, el proceso software como conjunto de herramientas, métodos y prácticas utilizados para producir software [5] posee unas características inherentes – Visión parcial, incertidumbre, interdependencia de componentes y comunicación informal – que requieren técnicas de coordinación que pueden no ser necesarias en otros entornos de producción más rutinarios [2]. Así, el proceso de coordinación se convierte en uno de los riesgos potenciales en la gestión del proyecto.

Por otra parte, la medición de la efectividad de un proceso posibilita que la gestión controle mejor los costes, reduzca riesgos y mejore la calidad. Además, la medición mejora la objetividad de la comunicación en cuanto a planes, status de desarrollo del proceso y, lo que es más importante, la efectividad del proceso software standard de la organización y la librería de información relacionada.

En resumen, la medición de la coordinación, proporcionará a gestión una mejor comprensión sobre la efectividad de los activos del proceso que nos ayudará a mitigar los riesgos del proyecto.

Nuestro objetivo en este artículo es determinar unas medidas prácticas básicas iniciales para el proceso de coordinación que propicien su estudio y análisis posterior. Se considera también importante proponer unas medidas no solamente teóricas, sino que en un futuro puedan implementarse con éxito en el mundo empresarial, para lo cual se considera igualmente importante determinar también los elementos de datos a recolectar.

Para conseguir este objetivo se realiza una propuesta para la medición práctica de la coordinación dentro del proceso software utilizando como guías las prácticas clave de los modelos de madurez CMM, el proceso de medición Goal-Questions-Indicators-Measures, más conocido como GQ(I)M, así como un software gratuito que implementa el proceso de medición PSM.

En cuanto a la estructura, el artículo consta 6 apartados cuyo contenido pasamos a detallar a continuación.

El primero describe la motivación y estructura del artículo. El segundo revisa los modelos de madurez CMM y CMMi en cuanto al área de proceso referente a la coordinación, establece una relación entre las actividades, así como los elementos del modelo. El tercero determina unas actividades de medición mínimas para coordinación basadas también en los modelos de madurez. El cuarto explica someramente los procesos de medición GQ(I)M y PSM, presenta el software gratuito PSM Insight y establece la relación existente entre las actividades del modelo CMMi y GQ(I)M. En el quinto apartado se utilizan los tres apartados anteriores para proponer

las medidas básicas, indicadores y atributos y elementos de datos a considerar en la medición del proceso de coordinación. Por último, en el sexto apartado, se establecen las líneas de trabajo futuras.

2. La coordinación en los modelos de madurez de la capacidad del software CMM[1]

Los modelos de madurez CMM se utilizan para el establecimiento y mejora de procesos en una organización midiendo la capacidad de la misma según una escala de cinco niveles que nos indican la madurez de sus procesos – inicial, repetible / gestionado, definido, cuantificado y optimizado.

Dichos modelos contienen los elementos esenciales de procesos efectivos para una o más áreas y representan una guía para definir y mejorar distintos procesos de la organización, que corresponden a distintas áreas.

El modelo CMMI – o CMM Integrado - constituye una evolución del estándar inicial para reunir en un solo modelo los distintos CMM que existían en ese momento (por ejemplo CMM-SW y CMM-SE).

Ambos modelos permiten dos tipos de representación: continua o por etapas. En la última de ellas se da un mapa predefinido, dividido en etapas (los niveles de madurez), para la mejora organizacional basado en procesos probados, agrupados y ordenados y sus relaciones asociadas.

Tanto en el modelo CMM como en CMMi, el tema de la *coordinación* se trata en el nivel 3 de madurez – Proceso definido –. Las *actividades* del proceso se denominan prácticas clave, describen las actividades que más contribuyen a la implementación eficiente de un área de proceso y debemos cumplirlas todas para la consecución de los objetivos del área de coordinación.

El proceso de coordinación en el modelo CMMI se encuentra como *parte* del área de proceso "Gestión Integrada del Producto" (PA IPM[2]), concretamente en el objetivo específico (SG 2) cuyas prácticas específicas (SP) para conseguirlo se detallan en la tabla 1.

SG 2 La coordinación y colaboración del proyecto se lleva a cabo con los interesados relevantes
SP2.1 Gestionar la implicación de los interesados relevantes en el proyecto.
SP2.2 Participar con los interesados relevantes para identificar, negociar y hacer un seguimiento de las dependencias críticas.
SP2.3 Resolver los problemas de coordinación con los interesados relevantes

Tabla 1. SG 2 y prácticas específicas para IPM

A diferencia del modelo CMMi, en el modelo CMM-SW existe un área clave de proceso (KPA) *específica* denominada "Coordinación Intergrupos" – KPA IC[3] - cuyas actividades (AC) se evidencian en la tabla 2.

Actividades de la KPA IC
IC AC-1 Participación para establecer los requerimientos a nivel de sistema. Todos los grupos participan.
IC AC-2 Monitorizar y coordinar las actividades técnicas. Todos los grupos participan.
IC AC-3 Comunicar los compromisos intergrupo, coordinar y hacer un seguimiento del trabajo desempeñado.
IC AC-4 Identificar, negociar y hacer un seguimiento de las dependencias críticas entre los grupos de ingeniería.
IC AC-5 Revisar los artefactos de trabajo recibidos para asegurar que cumplen con los requerimientos.
IC AC-6 Resolver las cuestiones intergrupo utilizando un procedimiento documentado.
IC AC-7 Llevar a cabo revisiones periódicas con la representación intergrupo.

Tabla 2. Actividades de la KPA IC

Sin embargo, existe una correspondencia entre ambos tipos de actividades, como queda reflejado en la tabla 3.

[1] Capability Maturity Model
[2] Process Area Integrated Product Management
[3] Key Process Area Intergroup Coordination

CMM - IC	CMMI - IPM
IC AC-1	IPM SP 2.1
IC AC-2	IPM SG 2
IC AC-3	IPM GP 2.2
IC AC-4	IPM SG 2, SP 2.2
IC AC-5	
IC AC-6	IPM SP 2.3
IC AC-7	IPM SP 2.1, SP 2.2, SG 2

Tabla 3. Correspondencia actividades IC con IPM

Las actividades y *artefactos* producidos en estas áreas de coordinación tienen que ver con los *procesos utilizados para coordinar y comunicar el esfuerzo de desarrollo*. Cada una de las prácticas clave de coordinación se particularizán en la organización bajo el contexto de cumplir los objetivos definidos para el área clave.

Para que el proceso de coordinación quede totalmente definido no basta con conocer las actividades y artefactos que lo componen, sino que existen otra serie de elementos esenciales, entre los que se encuentran: propósito y objetivos, participantes, criterios de entrada , criterios de salida (terminación), entradas al proceso y salidas del mismo.

El *propósito* de la coordinación intergrupos es el establecimiento de un método para que el grupo de ingeniería del software participe activamente con los otros grupos de ingeniería para que el proyecto sea capaz de satisfacer mejor las necesidades del usuario efectiva y eficientemente.

Los *objetivos* que se pretenden conseguir son Todos los grupos afectados acuerden - por escrito - los requerimientos del cliente y los compromisos intergrupo. Todas las cuestiones intergrupo están *identificadas, se hace un seguimiento y se resuelven.*

La coordinación intergrupos involucra la *participación* del grupo de ingeniería del software con otros grupos de ingeniería del proyecto para tratar los requerimientos, objetivos y cuestiones (issues) a nivel de sistema. Los representantes de los grupos de ingeniería del proyecto participan en el establecimiento de los requerimientos, objetivos y planes a nivel de sistema mediante trabajo con el cliente y los usuarios finales, cuando sea apropiado. Estos requerimientos, objetivos y planes se convierten en la base para todas las actividades de ingeniería.

Por último, las *entradas* al proceso las forman los compromisos y habilidades a alcanzar (por ejemplo, conseguir financiación y recursos adecuados). Ejemplos de *salidas* podrían ser agendas y calendarios para la actividades colaborativas.

Si se necesitase información adicional sobre las mediciones se podrían consultar las referencias sobre los modelos de madurez CMM [4]-[9].

3. La medición de la coordinación en los modelos CMM

En cuanto a la medición de la coordinación, en el modelo CMM-SW se incluye como una práctica clave adicional: (IC Meas 1) *Se realizan y utilizan mediciones para determinar el estado de las actividades de coordinación intergrupos*.
Ejemplos de mediciones son:

- Esfuerzo real y otros recursos gastados por el grupo de Ingeniería del Software para apoyar a otros grupos de Ingeniería.
- Esfuerzo real y otros recursos gastados por los otros grupos de Ingeniería para apoyar al grupo de Ingeniería del Software.
- La terminación real de tareas e hitos específicos del grupo de Ingeniería en apoyo de las actividades de otros grupos de ingeniería.
- La terminación real de tareas e hitos específicos de otros grupos de Ingeniería en apoyo del grupo de Ingeniería del Software.

Por el contrario, en el modelo CMMi, al integrarse la coordinación con la gestión del producto, las medidas necesarias deberán integrarse con las del proceso software. En este sentido, para la medición de un proceso software que se defina mediante los modelos de madurez CMM, la SEPO[4] del DoD propone en [11] las siguientes actividades mínimas:

M1: Medición de los Hitos de Desarrollo del Proceso. Se incluiría hacer un seguimiento de las actividades de desarrollo asociadas con el desarrollo y/o mantenimiento de los artefactos del proceso estándar. El desarrollo de las definiciones del proceso debería estar manejado con las mismas disciplinas formales involucradas en el desarrollo de cualquier producto final contractual.

[4] Software Engineering Process Office

Sin una visibilidad del status de las definiciones del proceso, están en riesgo la implementación y control global de los procesos organizacionales estándar. Cada artefacto debería estar sujeto a un calendario planificado (es decir, borrador, preliminar y final), y debería hacerse un seguimiento de ese plan.

M2: Efectividad del Proceso. Se incluiría una base de datos de estadísticas con el impacto de implementar un proceso dentro de un proyecto. La información baseline sobre los ratios de productividad y error de los proyectos que no utilizan procesos estándar necesita ser comparada con los resultados de proyectos que adapten los procesos estándar. De esta manera, puede evaluarse la efectividad de un proceso dado y puede ser presentada a otros en una base cuantificable.

M3: Utilización de Artefacto del Proceso. Se incluiría mantener un registro de la frecuencia de utilización de cada proceso estándar y de los items relacionados con el proceso a partir de la librería gestionada. Esto es importante para realizar un análisis tal como qué items están siendo adaptados, si deberían continuar en la librería o si un item debería hacerse más visible a los proyectos dentro de la organización.

Si se necesitase información adicional sobre las mediciones se podrían consultar las referencias sobre los modelos de madurez CMM citadas en el apartado anterior más [10] y [11]

4. La medición práctica del proceso software mediante GQ(I)M

El proceso de medición *GQ(I)M* es un proceso dirigido al objetivo que comienza mediante la identificación de objetivos de negocio y los divide sucesivamente en subobjetivos gestionables [1].

Se finaliza con la obtención de un plan para la implementación de medidas bien definidas e indicadores que soporten dichas medidas. A lo largo de todo el proceso se mantiene una trazabilidad con los objetivos, de manera que aquéllos que recojan y procesen los datos de medición no pierdan de vista dichos objetivos.

El proceso GQ(I)M consta de diez pasos sucesivos [4]:

1. Identificar requerimientos del negocio
2. Identificar qué es lo que queremos saber o aprender
3. Identificar subobjetivos
4. Identificar entidades y atributos medibles relacionados con los subobjetivos
5. Formalizar los objetivos de medida: Se ponen en contexto los objetivos de medida mediante propósito, perspectiva y entorno
6. Identificar las cuestiones cuantificables y los indicadores que se utilizarán para ayudarnos a conseguir los objetivos de medida.
7. Identificar los elementos de datos que recogeremos para construir los indicadores.
8. Definir las medidas e indicadores de la organización
9. Identificar las acciones que llevaremos a cabo para implementar las medidas.
10. Preparar un plan para implementar las medidas.

Estos pasos nos permiten hacer un seguimiento del desempeño software manteniendo en todo momento la correspondencia entre las mediciones realizadas y los objetivos de medida de nuestra organización.

Por otra parte, *PSM*[5] es un proceso de medición dirigido hacia la información, que trata los objetivos tanto de negocio como técnicos de una organización [12]. La orientación en PSM engloba las mejores prácticas realizadas por los profesionales de la medición dentro de las comunidades de ingeniería del software. Dicho proceso sigue los preceptos de GQ(I)M y dispone de un software gratuito para soportar su implementación: El PSM Insight. En este software se facilita el establecimiento de métricas mediante una guía en la que se relacionan las cuestiones (o áreas comunes de problemas), categorías y medidas (I-C-M). En la figura 1 podemos ver parte de esa guía.

Schedule and Progress	Milestone Performance	Milestone Dates Critical Path Performance
	Work Unit Progress	Requirements Status Problem Report Status Review Status Change Request Status Component Status Test Status Action Item Status
	Incremental Capability	Increment Content - Components Increment Content - Functions
Resources and Cost	Personnel	Effort Staff Experience Staff Turnover

Figura 1. La guía I-C-M de PSM

[5] Practical Software and System Measurement

5. Medidas prácticas propuestas para el proceso de coordinación

Cuando el objetivo inicial en el paso 1 de GQ(I)M es, por ejemplo, *conseguir una mayor comprensión del proceso de coordinación*, podemos establecer una correspondencia entre las cuestiones con los subobjetivos gestionables que tratamos de formalizar en el paso 5 del proceso GQ(I)M. Por otra parte, estos subobjetivos cuestionables se podrían enlazar con las actividades del proceso de coordinación

Así, para obtener unas *medidas iniciales* para el proceso de coordinación podríamos comenzar por el paso 5 del modelo GQ(I)M, que es precisamente lo que nos ayuda a implementar PSM Insight. La relación entre el Paso GQ(I)M y el elemento PSM Insight correspondiente se puede apreciar en la tabla 4.

Paso GQ(I)M	**PSM Insight**
Paso 5	Structure Measurement Plan Report (MPR)
Paso 6	Issue Indicator
Paso 7	Structure Data Item
Paso 8	Measure Data Item Attributes Indicator
Paso 9	MPR PSM Insight Import Export
Paso 10	MPR

Tabla 4. Correspondencia entre los pasos GQ(I)M y PSM Insight

Por otro lado, en el apartado anterior hemos indicado las medidas mínimas para el proceso software y las referentes al proceso de coordinación, así como el programa PSM Insight. Las áreas comunes en PSM Insight para dichas medidas se reflejan en la tabla 5.

Medida CMM	**Issue PSM Insight**
M1	Schedule and Progress
M2	Process Performance
M3, Ic Meas 1	Resources and Cost
IC Meas 1	Schedule and Progress Resources and Cost

Tabla 5. Correspondencia medida- cuestión

Es decir, para la medición de la coordinación, las cuestiones a abordar corresponderían a las áreas predefinidas "Calendario y Progreso", "Recursos y Coste" y "Desempeño del Proceso". Concretamente, las categorías y medidas relevantes serían:

Issue-Category-Measure para coordinación
Schedule and Progress Milestone Performance Milestone Dates Critical Path Performance
Resources and Cost Personnel Effort Environment and Support Resources Resource Utilization
Process Performance Process Efectiveness Defect Containment Rework

Tabla 6. Medidas PSM apropiadas

Tomando como base las actividades del área de proceso y considerándolas objetivos para el paradigma GQ(I)M se personalizarían para el proyecto concreto, siendo posible añadir, tanto más categorías como más medidas. Para poder observar el proceso de coordinación las mediciones se deberían realizar a nivel de actividad de coordinación y de grupo de trabajo.

En cuanto a los *indicadores*, se pretenderá que respondan a las preguntas surgidas de las prácticas clave de CMM. Por ejemplo, para medir el esfuerzo real vs planificado en las actividades de coordinación para cada puesto clave del proyecto, se podría realizar un gráfico de barras similar al de la figura 2.

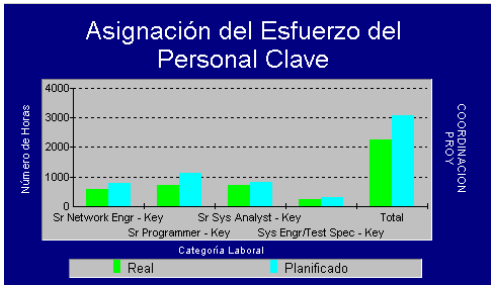

Figura 2. Ejemplo de indicador para coordinación

En cuanto a *qué nivel* (estructura), *atributos* y *elementos de datos* a recolectar, por ejemplo para la medida esfuerzo tendríamos:

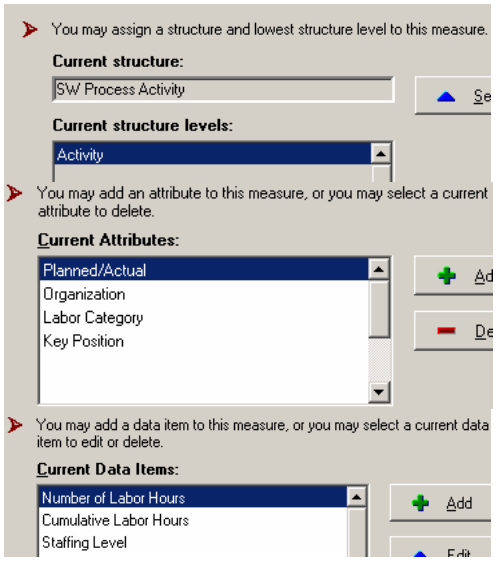

Figura 3. Definición de la medida esfuerzo

Para el resto de medidas encontradas los atributos y elementos de datos propuestos serían los que se citan a continuación.

Fechas de hitos

- Fecha
- Planificado/real
- Organización
- Versión
- Evento programado
- Fecha de comienzo
- Fecha de fin

Desempeño del camino crítico

- Elemento de configuración del software
- Fecha
- Planificado/real
- Organización
- Versión
- Evento programado
- Grado evento retrasado
- Causa de retraso en el calendario
- Fechas de comienzo
- Fechas de fin
- Actividad dependiente
- Tiempo para finalizar cada actividad
- Tiempo holgura para cada actividad

Utilización del recurso

- Fecha
- Planificado/real
- Organización
- Versión
- Recurso clave Sí/No
- Nº de horas requerido
- Nº de horas asignado
- Nº de horas programado
- Nº de horas disponible
- Nº de horas utilizado
- Nº de unidades requerido
- Nº de unidades asignado
- Nº de unidades programado
- Nº de unidades disponible
- Nº de unidades utilizado

Contención de defectos

- Fecha
- Actividad
- Organización
- Identificador del proceso
- Tipo de escape
- Nº de defectos detectados durante el proceso
- Nº de defectos detectados una vez que el proceso acabó.

Retrabajo

- Categoría Laboral
- Nº de horas labor retrabajadas
- Nº de componentes afectados por el retrabajo.

6. Líneas de trabajo futuras

En los apartados anteriores se realiza una propuesta para establecer medidas prácticas iniciales en el proceso de coordinación utilizando los estándares CMM y el paradigma GQ(I)M. También se detectan algunos indicadores y

elementos de datos a considerar. El siguiente paso sería realizar simulaciones, en la medida de lo posible con datos reales, para determinar si verdaderamente son adecuadas.

Agradecimientos

Esta investigación se ha realizado bajo el proyecto del Ministerio de Educación y Ciencia TIN2004-06689-C03-01.

Referencias

[1] Florac, A., Park, R.E. & Carleton A.D. Practical Software Measurement: Measuring for process management and improvement, Software engineering institute, guidebook CMU/SEI-97-HB-003, 1997, http://www.sei.cmu.edu/pub/documents/97.reports/pdf/97hb003.pdf (accedido Julio 2007)

[2] Kraut, R. E. & Streeter, L. A. Coordination in software development. Communications of the ACM 38(3): 69-81, 1995

[3] Park, R.E., Wolfhart, B., Geothert, W. and Florac, A., Goal-driven Software Measurement, software engineering institute, handbook CMU/SEI-96-HB-002, 1996 http://www.sei.cmu.edu//pub/documents/96.reports/pdf/97hb002.96.pdf (accedido Julio 2007)

[4] Paulk, Mark C. A History of Capability Maturity Model for Software. SEI Carnegie Mellon University http://www.sei.cmu.edu/cmm/slides/cmm-history.pdf (accedido Julio 2007)

[5] Paulk, Mark C., Curtis, Bill, Chrissis, Mary Beth & Weber, Charles V. Capability Maturity Model[SM] for software version 1.1 technical report CMU/SEI-93-TR-024 ESC-TR-93-177 February 1993

[6] SEI Software Engineering Institute http://www.sei.cmu.edu

[7] SEI CMMI for Software Engineering CMMI-SW V1.1 staged representation CMU/SEI-2002-TR-029 ESC-TR-2002-029

[8] SEI CMM Key Practices for Level 3 - Intergroup Coordination http://aelinik.free.fr/cmm/tr25_l3f.html (accedido Julio 2007)

[9] SEI CMMI-SE/SW v.1.1 to SW-CMM v1.1 STSC-mappings http://sei.cmu.edu/cmmi/adoption/pdf/stsc-mappings.pdf (accedido Julio 2007)

[10] SEPO (Software Engineering Process Office). SSC San Diego process asset library (PAL) http://sepo.spawar.navy.mil/

[11] SEPO An implementation guide to the SEI´s software capability maturity model key process areas: Integrated Software Management, Software product Engineering Inter-group coordination ISM/SPE/IC guide PR-ISM-02 v3.0, 2000 http://sepo.spawar.navy.mil/Process_Assets_By_KPA.html

[12] PSM Practical Software & Systems Measurement http://www.psmsc.com/

Estudio de los métodos de estimación: AHP y redes Bayesianas

Joseba Esteban López , José Javier Dolado
Deptartamento de Lenguajes y Sistemas
Universidad del País Vasco U.P.V./E.H.U.
jesteban004@ikasle.ehu.es , dolado@si.ehu.es

Resumen

Existen multitud de métodos de estimación para el cálculo del esfuerzo en el desarrollo del software. No obstante, los resultados de estos métodos no resultan satisfactorios o son demasiado dependientes del dominio del problema. En este artículo se exponen dos métodos que incluyen el conocimiento experto como base en sus estimaciones: AHP y redes Bayesianas.

1. Introducción

La estimación y medición desempeña un papel muy importante dentro de la creación y mantenimiento de productos y sistemas software. Las estimaciones y las medidas se suelen utilizar para evaluar la viabilidad de un proyecto, analizar sus alternativas, predecir el desarrollo y determinar la cantidad de recursos necesarios. Una buena estimación es clave, y se encuentra acotada por los costes de tiempo y dinero.

El aprovechar la experiencia anterior de proyectos software similares, contribuye a mejorar la exactitud de las estimaciones. Estas últimas suelen requerirse en las etapas iniciales del proyecto software. En estas fases iniciales los proyectos no suelen estar totalmente definidos y se van refinando según se va desarrollando el proyecto. La falta de disponibilidad de esta información previa incrementa la dificultad de conseguir buenas estimaciones, lo que a su vez provoca que los estimadores pasen por alto algunos componentes o actividades del proyecto, además de tener en cuenta supuestos inválidos. Es por esto que sería interesante poder realizar estimaciones en las fases iniciales del proyecto software, sin disponer de un gran número de datos previos y sin que se vea afectada la exactitud de las estimaciones.

Uno de los retos a los que se deben enfrentar los gestores, se produce a raíz de la aparición de cambios rápidos e impredecibles que afectan a sus estimaciones. Los cambios ocurren inevitablemente a lo largo del ciclo de vida del proyecto y en cualquier área del mismo. Por tanto, el equipo del proyecto se ve obligado a realizar continuas mediciones durante el ciclo de vida del proyecto. Comparar estas mediciones con los valores del objetivo del proyecto permite que el equipo del proyecto pueda lograr el objetivo. Los proyectos software tienen un grado alto de novedad, lo que los caracteriza como difíciles de estimar. La adaptación a los cambios no sería posible sin unas estimaciones lo suficientemente exactas. Unas estimaciones poco exactas o incorrectas afectarían a cualquiera que esté asociado al proyecto, desde los ingenieros y gestores, hasta los clientes, así como a la aceptación del producto, el éxito comercial, el coste operativo y la seguridad.

Hacer estimaciones lo suficientemente exactas conlleva un alto nivel de dificultad. A lo largo de los últimos treinta años, las técnicas de estimación han ido evolucionando, y se han desarrollado diferentes métodos de estimación. Analytic Hierarchy Process es un método de estimación que nos permite realizar estimaciones en fases muy tempranas del proyecto software y sin la necesidad de disponer de datos previos. Las redes bayesianas, en cam-

Definición	Explicación	Valor Realitvo	Valor Recíproco
Mismo tamaño	Las dos entidades tienen aproximadamente el mismo tamaño	1	1.00
Ligeramente mayor (menor)	La experiencia o el juicio reconoce una entidad como algo más grande (más pequeño)	3	0.33
Mayor (menor)	La experiencia o el juicio reconoce una entidad como definitivamente más grande (más pequeño)	5	0.20
Mucho mayor (menor)	El dominio de una entidad sobre otra es evidente; un diferencia muy fuerte de tamaño	7	0.14
Extremadamente mayor (menor)	La diferencia entre las entidades comparadas es de un orden de magnitud	9	0.11
Valores intermedios entre puntos adyacentes de la escala	Cuando el compromiso es necesario	2, 4, 6, 8	0.5, 0.25, 0.16, 0.12

Cuadro 1: Escala verbal propuesta por Thomas L. Saaty

bio, poseen la cualidad de adaptarse a las posibles alteraciones en el proyecto gracias a la realimentación de sus prioridades a posteriori. En este artículo se presentan los dos métodos de estimación: AHP y Redes Bayesianas.

2. Métodos de estimación: AHP y Redes Bayesianas

En esta sección se muestran dos métodos de estimación. Se señalan tanto sus características más importantes como su funcionamiento, así como las características de cada uno de ellos.

2.1. Analityc Hierarchy Process

A pesar de la cantidad de métodos de estimación existentes para prever el tamaño y el coste del software, la mayoría de los gestores de proyectos siguen utilizando los juicios expertos en sus estimaciones. El hecho de que los responsables de la gestión de los proyectos sigan desconfiando de los métodos de estimación puede deberse a diferentes causas, como la falta de información en las fases iniciales de los proyectos software, la especificidad del dominio sobre el que trata el software o el nivel de esfuerzo y tiempo requeridos por determinados métodos de estimación. Por el lado contrario, las estimaciones basadas únicamente en juicios expertos suelen aquejar de falta de exactitud (cómo de cerca está la medida del valor real).

Analytic Hierarchy Process (A.H.P.) es un método de estimación de ayuda a la toma de decisiones basado en múltiples criterios de decisión. AHP fue propuesto por Thomas L. Saaty en la década de los 80. Desde entonces se ha convertido en una de las técnicas más utilizadas para la toma de decisiones multiatributo. AHP se basa en juicios subjetivos realizados por los expertos. Los expertos aportan su conocimiento subjetivo, consistente en comparaciones entre las principales tareas que constituyen un proyecto software. Los expertos estiman, más que un valor exacto, una medida relativa. Basándose en esta idea, el experto, evaluando la proporción entre cada par de tareas definidas en la aplicación software, consigue una mayor exactitud en sus evaluaciones.

2.1.1. Algoritmo AHP

El algoritmo del método de estimación Analytic Hierarchy Process consta de cinco pasos que exponen a lo largo de este apartado.

En primer lugar se define el problema. Para esto hay que dividirlo en tres partes: objetivo, criterios y alternativas. El objetivo es la decisión que se ha de tomar. Los criterios representan los factores que afectan a la preferencia o deseabilidad de una alternativa. Pueden estar compuestos por otros criterios o subcriterios. Las alternativas son las posibles opciones o acciones de las que se dispone y de las cuales se intenta elegir una. Una alternativa puede ser cualquier entidad relevante en un grupo de interés, como casos de uso, módulos soft-

$$A^{n\times n} = \begin{cases} a_{ij} = \frac{s_i}{s_j} & \text{Cómo de grande o pequeña es la entidad} i \text{respecto de la entidad} j \\ a_{ij} = 1 & \text{Las entidades} i y j \text{son de la misma proporción} \\ a_{ij} = \frac{1}{a_{ji}} & \text{Inversamente proporcionales} \end{cases} \quad (1)$$

ware, objetos etc., es decir, cualquier entidad de la que se pueda conocer las magnitudes que se necesitan a la hora de tomar una decisión. Una vez hecho esto, se debe construir la jerarquía, de la que AHP toma el nombre. Se han de tomar una serie de decisiones identificando qué factores son importantes y cómo interactúan con el resto de factores.

Aunque no se trate de una parte esencial de la metodología de AHP, establecer una escala verbal o *verbal scale* agiliza el proceso de estimación y no hace peligrar la exactitud de la estimación. La escala verbal ayuda a entender cómo de menor es el término "menor que"ó cómo de mayor es el término "mayor que". Se compone de cuatro atributos: definición o etiqueta, explicación, valor relativo y valor recíproco. La escala verbal establece un consenso que evita que los expertos o los participantes en la estimación pierdan tiempo discutiendo sobre el grado de diferencia entre las alternativas comparadas.Thomas L. Saaty [1] nos propone una escala compuesta por 9 valores y sus recíprocos, como se puede ver en el Cuadro 1.

Con la jerarquía y la escala verbal ya bien definidas, se pasa obtener la matriz de juicios o *judgement matrix*. Es en esta etapa donde entra en juego el juicio experto. El experto, basándose en la escala verbal, debe hacer comparaciones por parejas en cada nivel de la jerarquía, e ir anotándolos en la matriz. La matriz de juicios es de tamaño nxn, siendo n el número de alternativas de las que se dispone. Cada celda de la matriz de juicios contiene un valor a_{ij}, que representa el tamaño relativo de la entidad i respecto del de la entidad j. Los elementos de la matriz se definen como se muestra en (1). Si la entidad i es a_{ij} veces mayor (o menor) que la entidad j, entonces la entidad j es $1/a_{ij}$ veces menor (o mayor) que la entidad i. Teniendo en cuenta esta premisa y que la diagonal de la matriz sólo tiene como valor la unidad, no haría falta calcular todos los valores de la matriz. Sólo es necesario calcular una mitad de la matriz, ya sea la parte superior a la diagonal o la inferior.

A la hora de hacer las comparaciones por parejas, es necesaria la colavoración del experto y al menos una entidad de referencia o *reference task* de la que se conozca su magnitud real de proyectos anteriores. Las proporciones de la entidad de referencia son las primeras que se han de situar en la matriz. Es importante que la proporción de esta entidad no ocupe los valores extremos de la escala verbal, sino que se sitúe más o menos hacia la mitad de la escala. De esta manera se minimizan los posibles prejuicios introducidos en la matriz de juicios. Otra posibilidad, con el mismo objetivo, radica en introducir más de una entidad de referencia repartidas uniformemente en la escala verbal.

En este punto, con una matriz de juicios por cada criterio a tener en cuenta en la toma de decisión, se calcula la escala de proporción o *ratio scale*. La escala de proporción es un vector r en el que cada posición del vector contiene un valor proporcional a la entidad i en relación al criterio elegido. También se calcula el índice de inconsistencia o *inconsistency index*. Este índice nos proporciona una medida de cómo de lejos está nuestra estimación de la consistencia perfecta. Una matriz de juicios perfectamente consistente es aquella en la que todos sus elementos satisfacen $a_{ij} \times a_{jk} = a_{ik}\ \forall i,j,k$. Como procedimiento para calcular la escala de proporción y el índice de inconsistencia, proponemos la utilización del modelo propuesto por Eduardo Miranda en [1], por sencillez y buenos resultados. Hay que calcular la media geométrica v_i de cada fila de la matriz de juicios definida para un determinado criterio. El valor de v_i viene dado por (2). La escala de proporción,

denominada vector de valores propios o *eigenvalue*, r, consiste en un vector en el que cada valor se calcula aplicando (3).

$$v_i = \sqrt[n]{\prod_{j=1}^{n} a_{ij}} \quad (2)$$

$$r = [r_1,\ r_2,\ \ldots\ , r_n]\ con\ r_i = \frac{v_i}{\sum_{j=1}^{n} v_j} \quad (3)$$

El índice de inconsistencia se puede calcular de la siguiente forma (4).

$$CI = \frac{\sqrt{\sum_{i=1}^{n}\sum_{j>i}^{n}\left(ln\ a_{ij}\ -\ ln\ \frac{v_i}{v_j}\right)^2}}{\frac{(n-1)\times(n-2)}{2}} \quad (4)$$

A partir del vector de valores propios o *eigenvalue* y el valor real de la entidad de referencia, se puede calcular el valor absoluto de cada entidad. Para ello se debe aplicar la expresión (5). Aplicando estos cálculos con el resto de criterios, previamente ordenados según el porcentaje de contribución sobre el proyecto en su totalidad, se puede calcular el valor de cada alternativa.

$$Valor_i = \frac{r_i}{r_{refrencia}} \times Valor_{refrencia} \quad (5)$$

En el siguiente apartado, se muestran las características más relevantes de AHP . También se indican los principales problemas que presenta este método de estimación.

2.1.2. Características de AHP

Analytic Hierarchy Process es uno de los métodos de estimación más sencillos cuya mayor dificultad radica en identificar los atributos y su contribución relativa. Además, proporciona una visión del proyecto software jerarquizada, estructurada y sistemática. Asimismo, AHP es poco propenso a errores, permitiendo estimaciones precisas con hasta un 40 % de comparaciones erróneas. A pesar de este dato no se puede afirmar que AHP sea mejor que la estimación experta, ya que está basado, precisamente, en comparaciones hechas por expertos entre pares de tareas.

AHP aporta una notable ventaja para los expertos, ya que resulta más sencillo hacer comparaciones por parejas (entre los pares de tareas) que estimar cada tarea de una en una. Por otro lado, el número de comparaciones que deben realizar puede suponer un problema. Esto se debe a que la relación de las comparaciones a realizar y el número de tareas es de orden cuadrático. Si nuestro proyecto se compusiera de n tareas, el número de comparaciones que se deberían realizar sería de $n(n-1)/2$. En el supuesto de manejar 30 tareas, se deberán realizar 435 comparaciones. Para evitar esto, aún a riesgo de aumentar la homogeneidad de la matriz de juicios, se pueden agrupar las tareas similares en grupos más reducidos.

En las fases iniciales del desarrollo de un proyecto software suele darse una carencia de datos de referencia. El método de estimación AHP resulta muy útil en estas etapas iniciales, ya que como mínimo necesita un único dato de referencia: la tarea de referencia. Esta característica nos posibilita hacer estimaciones bastante precisas en fases tempranas del desarrollo de un proyecto software. Es importante que el porcentaje de contribución de la tarea de referencia al proyecto global sea lo más preciso posible. Esto aumentará la exactitud de las estimaciones del resto de las tareas. En caso contrario, podrá derivar en errores. La exactitud de las predicciones también se verá mejorada con el uso de más de una tarea de referencia.

Con el método de estimación AHP puede darse el fenómeno del Rank Reversal o alteración del rango. Este fenómeno consiste en un cambio en el ranking relativo de las tareas, al introducir una nueva tarea o eliminar una de ellas. Los autores Ying-Ming Wang y Taha M.S. Elhag en [2], exponen la existencia de varias propuestas enfocadas a evitar el fenómeno de la alteración en el rango. Estas propuestas tienen en cuenta si la alternativa introducida o eliminada del conjunto de alternativas seleccionadas agrega o no información. En ese mismo artículo se expone la propuesta de los autores, en la que se preserva el ran-

king de las alternativas sin necesidad de variar los pesos de las alternativas ni el número de criterios. Para esto, los autores proponen la normalización del vector de valores propios o *eigenvalue*.

Un aspecto a tener en cuenta utilizando el método de estimación AHP radica en la escala elegida, tanto por su proporción como por el número de puntos que la constituyen. En AHP el éxito en las estimaciones reside en la exactitud de las comparaciones entre las tareas. Estas comparaciones, son consecuencia directa de la escala de evaluación elegida, así como del número de puntos de la escala. La escala propuesta por Thomas L. Saaty en [1], propone el uso de una escala que varía entre 1/9 y 9, lo que supone un total de 17 puntos en la escala. En [3] los autores aconsejan el uso de una escala con un número de etiquetas más bajo, ya que esto nos facilitará el manejo de la escala. Eduardo Miranda en [1] también propone una escala diferente. Ésta consta de menos puntos que la propuesta por Saaty, y según el autor, es más fácil de utilizar por los expertos.

2.2. Redes Bayesianas

En este estudio se incluyen las redes Bayesianas dada su cada vez mayor popularidad dentro de la ingeniería del software, habiendo sido utilizadas en diferentes áreas como la estimación del esfuerzo y calidad, pruebas de software, fiabilidad e interfaces gráficas. Las redes Bayesianas son modelos gráficos probabilísticos que representan una función de distribución conjunta sobre un conjunto finito de variables. Formalmente, una red Bayesiana es un grafo dirigido acíclico.

El gráfico está dividido en una parte cualitativa y otra cuantitativa. La parte cualitativa, es el grafo en sí. El grafo está compuesto de nodos y arcos entre los nodos. Los nodos representan variables aleatorias del dominio $X_1,\ X_2,\ \ldots\ , X_n$, mientras que los arcos representan valores de dependencias entre las variables. Las redes Bayesianas asumen que un nodo depende únicamente de sus padres.

La parte cuantitativa representa la incertidumbre del problema por medio de probabilidades condicionadas: posibles relaciones causa efecto entre los nodos. Cada nodo posee una tabla de probabilidades condicionales asociada, que define la probabilidad de cada uno de los estados en los que puede estar una variable, dados los posibles estados de sus padres. Esta tabla de probabilidades representa los posibles valores (estados) que puede tomar esa variable y la probabilidad de que se dé cada uno de ellos. Una red Bayesiana muestra la probabilidad de distribución conjunta para un grupo de variables $X_1,\ X_2,\ \ldots\ , X_n$ (6), tal que el valor que toma la variable X_i viene representado por x_i. Los valores que tienen el conjunto de los padres de la red Bayesiana del nodo X_i se representa con $padres(X_i)$. Cada estado de una variable se calcula multiplicando un número de valores en las tablas de probabilidades condicionales.

$$P\left(X_1,\ X_2,\ \ldots\ , X_n\right) = \prod_{i=1}^{n} P\left(x_i | padres\left(X_i\right)\right) \tag{6}$$

Las redes Bayesianas se utilizan en problemas donde no se tiene un completo conocimiento del estado del sistema. Sin embargo, permite realizar observaciones con el fin de actualizar el resto de probabilidades. Cuando se conoce qué valor toma una variable se está hablando de evidencias. Existen dos tipos de evidencias: *firme* o *específica*, y *parcial* o *virtual*. La primera de ellas se da cuando se asigna un valor concreto a una variable. De los posibles valores o estados que puede tomar una variable, cuando uno de ellos tiene un probabilidad del 100 %, se denomina *certeza absoluta*. Las evidencias *parciales* o *virtuales* permiten actualizar las probabilidades a priori de los estados que puede tomar una variable, es decir, antes de tener la certeza absoluta de qué valor toma dicha variable.

Las variables continuas en las redes Bayesianas deben ser adaptadas al hecho de que cada variable sólo dispone de un conjunto de estados finitos en las tablas de probabilidad condicional. Una manera de abordar este problema radica en discretizar las variables continuas. La discretización consiste en dividir el rango de las variables continuas en

intervalos exhaustivos y exclusivos. Esto conlleva una perdida de información dependiente del dominio y del número de intervalos en los que se divide el rango de la variable continua.

Las probabilidades antes de introducir evidencias en la red Bayesiana, se conocen como *probabilidades a priori*. A partir de las evidencias de que se dispongan, junto con la red Bayesiana bien definida, se pueden calcular o estimar las variables de las que se desconocen sus valores. Una red Bayesiana proporciona un sistema de inferencia, donde una vez encontradas nuevas evidencias sobre el estado de ciertos nodos, se modifican sus tablas de probabilidad condicional y a su vez, las nuevas probabilidades son propagadas al resto de los nodos. Esta propagación de probabilidades recibe el nombre de *inferencia probabilística*. Las nuevas evidencias propagadas se llaman *probabilidades a posteriori*. Ante redes formadas por un gran número de nodos y dependencias, la propagación de probabilidades tiene un coste computacional alto, siendo un problema *NP-complejo*. Actualmente existen algoritmos de propagación de probabilidades con los que es posible modelar redes Bayesianas de problemas reales. Existen dos grandes grupos de algoritmos de propagación: los algoritmos de propagación exactos, cuando no hay error en las probabilidades calculadas, y los algoritmos aproximados, cuando las probabilidades de los nodos son estimadas con cierto margen de error.

2.2.1. Creación de redes Bayesianas

Para crear una red Bayesiana que modele un problema, básicamente hay que definir la estructura (grafo) y los parámetros de la red (tablas de probabilidad). Para esto, hay que seguir los pasos situados bajo estas líneas y realizar varias iteraciones de los mismos.

- Lo primero que se debe hacer, una vez que se tenga pleno conocimiento del dominio, es seleccionar las variables de interés. En dominios complejos puede resultar complicado enumerar todas las variables importantes y conocer sus relaciones causales.

- Una vez seleccionadas las variables de interés, se decide si cada variable del dominio será utilizada como variable de entrada, su tipo y si hay que discretizarla o no.

- El siguiente paso consiste en definir la topología de la red, es decir las relaciones causales entre las variables. Diseñando la topología de la red Bayesiana de esta manera se minimiza el número de arcos innecesarios y se maximizan las independencias condicionales.

- Para terminar de crear la red Bayesiana, hay que definir las tablas de probabilidad para cada uno de los nodos de la red. Los ingenieros del conocimiento pueden ayudarse de la minería de datos si existen bases de datos del dominio.

- Una vez creada la red Bayesiana es necesaria su evaluación y verificación de su utilidad. Con este propósito se utiliza el análisis de sensibilidad, que permite comprobar cómo varían los resultados de algunas variables ante cambios en los valores de algunas evidencias de ciertas variables.

A continuación se presenta un ejemplo de red Bayesiana sencilla, con el fin de mostrar el aspecto de las redes Bayesianas. En la (Figura 1) se puede ver la arquitectura de una red Bayesiana para la estimación del esfuerzo software sacado de [4], donde aparece explicado con más detalle. En este caso el nodo denominado esfuerzo software representa la variable dependiente. Este nodo depende de tres nodos padres: metodología de desarrollo, tipo de herramienta CASE y experiencia con las herramientas CASE. En esta figura se muestran los posibles estados en los que se puede encontrar cada variable y su distribución de probabilidad a priori. En el caso del nodo de esfuerzo software se muestra la tabla de probabilidades condicionadas. La distribución de probabilidades a priori se ha obtenido a partir de un conjunto de datos consistente en 33 proyectos software reales de dos grandes compañías del nordeste de Estados Unidos.

	Tipo Herramienta	INCASE						IEF					
	Experiencia	Alta		Media		Baja		Alta		Media		Baja	
	Metodología	NoRAD	RAD	NoRAD	RAD	NoRAD	RAD	NoRAD	RAD	NoRAD	RAD	NoRAD	RAD
Esfuerzo Software	Alto	0.0	-	0.0	0.0	0.06	-	-	-	0.0	-	1.0	0.0
	Medio	0.5	-	0.0	0.0	0.06	-	-	-	1.0	-	0.0	1.0
	Bajo	0.5	-	1.0	1.0	0.88	-	-	-	0.0	-	0.0	0.0

Metodología	Probabilidad
SDLC(no RAD)	0.788
RAD	0.212

Herramienta	Probabilidad
INCASE	0.849
IEF	0.151

Experiencia	Probabilidad
Alto	0.606
Medio	0.303
Bajo	0.091

Figura 1: Ejemplo de arquitectura de una red Bayesiana

2.2.2. Características de las redes Bayesianas

La aplicación de las redes Bayesianas en la ingeniería del software ha sido menor que en otras áreas como la medicina. Esto puede deberse a varias causas, como que el tamaño de las bases de datos en ingeniería del software sean más pequeñas, las mediciones sean más subjetivas, las herramientas estén madurando y no sean específicas de la ingeniería del software y, además, que las relaciones causa efecto dentro de los proyectos software sean difíciles de evaluar.

A pesar de esto, las redes Bayesianas poseen una serie de características idóneas para la ingeniería del software. Las redes Bayesianas proveen de una representación gráfica de las relaciones explícitas de dependencia del dominio, de manera que permiten modelar sistemas complejos posibilitando entender las relaciones causales visualizándolas por medio del grafo. Además, están compuestas por dos partes bien diferenciadas, la parte cualitativa y la cuantitativa, que permite utilizar criterios objetivos (otros proyectos finalizados) y subjetivos (la experiencia de los expertos del dominio), respectivamente. Por otro lado, las redes Bayesianas permiten la inferencia bidireccional, con lo se permite hacer inferencia en ambos sentidos. De esta manera se puede predecir que variables de entrada son necesarias para obtener las variables de salida deseadas. También permiten el uso de valores con grados de certidumbre. Esto permite modelar la incertidumbre de manera efectiva y explicita, por lo que se pueden generar buenas predicciones con información incompleta. Asimismo, la salida que proporciona una red Bayesiana es una probabilidad de distribución en vez de un valor concreto. Este tipo de información es útil a la hora de medir la confianza que se puede depositar en la salida de la red, y más si el modelo va a ser utilizado en la toma de decisiones. No se debe olvidar las facilidades que aportan las redes Bayesianas ante un análisis de sensibilidad, ya que permiten calcular la sensibilidad de ciertas variables de la red, simplemente modificando las evidencias.

Existen una serie de características que suponen una dificultad o limitación a la hora de crear redes Bayesianas. La definición de la estructura de la red Bayesiana puede suponer una tarea compleja incluso para expertos del dominio, pudiendo crear desigualdades entre el modelo construido y el problema del dominio. También se da el problema de que el modelo construido no tenga en cuenta una serie de relaciones poco evidentes que sí estén en el problema del dominio. Además, una mala estimación de los parámetros puede distorsionar toda la red invalidando los resultados. Las redes Bayesianas son sensibles a la inconsistencia, y sólo resultan útiles si se puede confiar en sus resultados.

Normalmente las estimaciones realizadas mediante redes Bayesianas en las fases iniciales del proyecto software raramente son correctas. Esto es debido a que en estas fases tempranas del desarrollo del proyecto software se cuenta con una definición pobre del proyecto, incluso a la hora de decidir que herramientas o metodologías utilizar. Las estimaciones con redes Bayesianas van mejorando durante el ciclo de vida del proyecto software.

3. Conclusiones

En este artículo se ha realizado un estudio de dos métodos de estimación del esfuerzo de desarrollo de software: AHP y redes Bayesianas. En las etapas iniciales de ciclo de vida del proyecto software ambos modelos se ven afectados por la falta de datos históricos. Con AHP se pueden realizar estimaciones precisas con sólo un dato de referencia, mientras que las redes Bayesianas normalmente no son capaces de proporcionar buenas estimaciones en estas fases tempranas del desarrollo del proyecto software. Una vez avanza el proyecto a lo largo del ciclo de vida, las estimaciones proporcionadas mediante redes Bayesianas mejoran notablemente. Ambos métodos se basan en el conocimiento experto: AHP a la hora de realizar las comparaciones y las redes Bayesianas en el diseño de éstas. Ambos métodos han demostrado su eficacia a la hora de estimar diversos tipos de problemas.

Agradecimientos: Este trabajo se ha desarrollado gracias a la financiación del proyecto TIN2004-06689-C03-01.

Referencias

[1] Eduardo Miranda. Improving subjective estimates using paired comparisons. *IEEE Software*, 18(1):87–91, Jan/Feb 2001.

[2] Ying-Ming Wang and Taha M. S. Elhag. An approach to avoiding rank reversal in ahp. *Decis. Support Syst.*, 42(3):1474–1480, 2006.

[3] Sahrman Barker Martin Shepperd and Martin Aylett. The analytic hierarchy process and data-less prediction. *Empirical Software Engineering Research Group ESERG: TR98-04*, 1998.

[4] Girish H. Subramanian Parag C. Pendharkar and James A. Rodger. A probabilistic model for predicting software development effort. *IEEE Transactions On Software Engineering*, 31(7):615–624, July 2005.

[5] Daniel Rodriguez and Javier Dolado. *Redes Bayesianas En La Ingeniería del Software*, pages 1–19. To appear in 2007.

[6] Richard D. Stutzke. *Estimating Software-Intensive Systems: Projects, Products, and Processes.* The SEI Series in Software Engineering. Addison Wesley Professional, http://sw-estimation.com/index.html, 1 edition, Apr 26 2005.

Overview of XBRL technologies for decision making in Accounting Information Systems

Eva Reyes
Buckinghamshire County Council, UK
erhernando@buckscc.gov.uk

Daniel Rodríguez
Universidad de Alcalá
Depto. de Ciencias de la Computación
28805 - Alcalá de Henares, Madrid, Spain
daniel.rodriguezg@uah.es

Javier Dolado
Depto. de Lenguajes y Sistemas Informáticos
Universidad del País Vasco, Facultad de Informática
20.009 Donostia - San Sebastián, Spain
dolado@si.ehu.es

Abstract

XBRL (eXtensible Business Reporting Language) is a language for the electronic communication of business and financial data based on XML (eXtensible Markup Language). Compared with paper based or other previous ad-hoc EDI (Electronic Data Interchange) technologies, XBRL provides major benefits in the preparation, analysis and communication of business information. Those benefits include cost savings, greater efficiency, accuracy and reliability to all activities involved in supplying or using financial data. This paper provides an overview of XBRL technologies and how it is applied to decision making in several financial areas. It also covers some possible extensions with the semantic Web and Web services as future challenges.

Keywords: XBRL, accounting, decision making, accounting information systems

1. Introduction

Since the inception of the Internet, many technologies have been proposed as EDI (Electronic Data Interchange) enablers to move information between systems. Currently, how quickly and how much information a company can obtain improves the decision making process to achieve greater efficiency or advantageous position against competitors. For example, investors wanting to use their own tools using data providers or accumulators as Bloomberg need to copy/paste such information manually, moving information manually from organization systems to authorizes systems, manager compiling information from different department into a spreadsheet in a consistently for decision making. However, the format of the information was different among the different systems. As a result, in the financial domain far more time was needed producing the information and getting it ready for analysis than into the actual analysis. With the creation of XBRL (eXtensible Business Reporting Language), there is another way.

XBRL is a language for the electronic communication of business and financial data. XBRL is an open and freely available standard language based on XML (eXtensible Markup Language)[1] for creating business reports. As a language, it does not intent to modify any of the GAAP (Generally

[1] http://www.w3.org/XML/

Accepted Accounting Practices) but to represent them. It can contain both financial information (e.g. balance sheets, income statements or cash flows) and non-financial information (e.g. performance measurements and statistics, loan applications or regulatory reporting forms). The XBRL is an open standard which can facilitates many of the activities in the Corporate Reporting Supply Chain (CRSC) that need financial or statistical data to be stored, exchanged and analysed such as reporting about company's financial status to all types of regulators and tax authorities, applications to banks and governments, risk assessments, etc. Mayor benefits include: faster time and cost of producing and accessing reports, elimination of need for porting, translating or mapping data, enable search and comparison of business data.

From the technical point of view, XBRL is replacing other previously defined XML standards for describing financial contents and business processes during the past few years such as FpML, RIXML, or ebXML. A reason for this, it is its wide support. XBRL International[2], a not-for-profit consortium of around 450 companies and agencies, is responsible for advancing this technology. Many regulatory authorities are recommending its use, for example, the US SEC (Securities and Exchange Commission) has modernised its database, called EDGAR, to support XBRL provides. EDGAR provides access to global company information from over 13,000 companies. Spanish CNMV (Spanish's national market supervisor), UK Inland Revenue, Australia Tax Office, Japan's Tax Agency and many others accept XBRL reports or will make it compulsory. Therefore many companies and stock exchanges use XBRL in their databases and as part of their external reporting systems. Also, it is supported by mayor software vendors such as SAP, Microsoft, Oracle, etc.

Furthermore, in the European Union, the Committee of Eurpean Baking Superviors[3] considers the development of XBRL as utterly important with the development of two taxonomies: the COREP[4] (Common solvency ratio Reporting) and FINREP (Financial Reporting) taxonomy which serves as a common reporting framework for financial data.

This paper is organized as follows. Setion 1 provides an overview of XBRL. Section 2 describes how XBRL capabilities are being used by in projects or tools to support decision making. Next, related technologies for XBRL are described. Finally, conclusions and future work are outlined.

2. XBRL (eXtensible Business Reporting Language) in a Nutshell

As stated previously, XBRL is an open standard based on XML and related standards (XML Schema, XML Namespaces and XLink) designed to share financial information and avoid previously problems with incompatibles types. It is, therefore, independent of any hardware or software platform.

XBRL is composed of a specification about how to structure business data and a common framework for structuring and naming business information. XML provides the structure for the data and different taxonomies created name, define, relate and classify different business concepts using tag lists. As an example, XBRL GL, the *journal taxonomy*, offers the representation of data for the general ledger and sub-ledger. It is worth noting that the taxonomy does not include actual data for the concepts, data is included in the XML instance documents as shown in Figure 1. XBRL taxonomies are created using XML Schema which is used to described the meaning of XML elements. The reason to

[2] http://www.xbrl.org/

[3] http://www.c-ebs.org/

[4] http://www.corep.info

create the different taxonomies is that different companies from various industry sectors describe terms differently, groups or task forces from are agreeing on the most relevant and useful vocabulary specific to their sector.

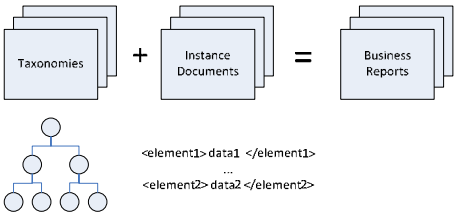

Figure 1 XBRL reports

We briefly explain the different technologies in the following items.

- XML (eXtensible Markup Language) describes data as elements between tags in plain text. For example, the balance of an account can be represented as <balance>545</balance>; element names are embraced by angle brackets (<…>), and the content (545) is between the opening element tag and the closing tag. Elements enclose other elements in a hierarchical way following a tree structure with a single root.
- XML Schema[5] documents define the structure of XML documents, i.e., its vocabulary. It defines which elements can appear, order in the hierarchy of the elements, its order and number, data type, default values and fix values. XML Schemas also make use of XML Namespaces to differentiate equal tag names with different meanings. For example, <title> can be prefixed with a namespace to differentiate book <book:title> or CD <cd:title>. A XBRL taxonomy is XML Schema that defines the concepts for a business reports and their relationships, for example, commonly used data types defined include: monetaryItemType, sharesItemType, decimalItemType, stringItemType, uriItemType, dateTimeItemType and tupleType.
- XML Linking (XLink)[6]. XBRL makes use of Linkbases to provide further meaning to concepts and to define relationships between concepts. With the Schema we defined concepts but balance sheets, assets, current assets, non-current assets, liabilities, etc. but we also need how know that current assets is a kind of asset, how to calculate assets from current and non-current assets or how to report these concepts. XBRL uses XLinks for 5 different purposes: (i) definition links, (ii) calculation links, (iii) presentation links, (iv) labels and (v) references. Definition links are used to describe relationships between concepts in a taxonomy; calculation links describes how elements are calculated; presentation links define relations needed for presentation, for example parent-child relationships. Labels and references do not define relationships but for *human* consumption. Labels are used to relate readable text with concepts and references to any kind of authoritative literature. These last two can also do not need to be in English, giving support for other languages.
- Instance Documents. They contain the actual facts for business reports, i.e., they represent the set of values of a taxonomic at one instance. There are usually several instances for one taxonomy. For example, there is one taxonomy for balance sheets but a company could have a balance sheet instance for every year. Concepts defined in the taxonomies, they are organised either as *items* or *tuples*. Items are basic data to be reported (e.g., <assests …>124</assets>); XRML taxonomies reference XML Schema simple and complex types and define the set of possible data an item can hold. On the other hand, *tuples* are concepts

[5] http://www.w3.org/XML/Schema

[6] http://www.w3.org/XML/Linking

that are used to contain other concepts, i.e., items or other tuples; for instance, the *address* can be composed of *street*, *city* and *postal code*. In addition to actual fact, instance documents contain other elements that represent contextual information (*context* elements) such as dates, duration or periods. Finally, other possible elements include *Unit* which represent units of measured items, *scenarios* to indicate circumstances of reported items, *precision* indicates how many digits can be trusted, footnotes, CWA (Closed World Assumption) to indicate whether the report is complete and if the information is valid and *group*s to arrange items in instance documents.

Sometimes, it is necessary create new taxonomies extending existing ones. For example, if a government or accounting standard has developed a taxonomy, a company would probably like to extend the jurisdictional taxonomy by adding its specific elements modifying relationships to the company financial reporting taxonomy. For example, jurisdictional taxonomy defines assets but a company may want to distinguish between cash in the bank and cash in the stock market.

3. Decision Making with XBRL

Tools used for improving the decision making processes in the financial domain include: Business Activity Monitoring, Digital Dashboard (also known as balanced scorecard systems), Portals, Business Intelligence tools and Data Warehousing and Data Mining. New generation of these tools are adopting XBRL as a result of a great variety of projects and organizations supporting it.

An important improvement is for auditors, as XBRL enables continuous auditing. XBRL allows auditors to generate reports within a much shorter timeframe than under the traditional model, which is called continuous auditing. As stated previously, one of the requirements when designing XBRL is its ability to collect organize, analyse and maintain information about business entities. Short timeframes when auditing can potentially trigger corrective actions in a much more useful way than traditional audits with longer time span between the analysis of the data and the reports. A project at the Emporia State University provides an on-line demonstration of such capabilities[7].

An example of project, the MUSING project[8], as stated in their Web site, delivers next-generation knowledge management solutions and services to enable perceptive business intelligence activities, directly at the End-Users premises. MUSING provides services for three application areas: (i) financial risk management, (ii) internationalization, and (iii) IT Operational risk and business continuity. To do so, the aim of this project is to develop Business Intelligence tools and modules based on the semantic Web and content systems for decision-making. The technologies needed for these types or tools are described in the next sections.

The Dutch National XBRL project[9] (Het Nerlandse Taxonomie Project), which is supported by Dutch ministries of Justice and Fianance, also provides on-line examples of capabilities of XBRL[10].

The Enhanced Business Reporting Consortium (EBR)[11] is an independent organization whose mission is to improve the quality, integrity, and transparency of information used for decision-making. XBRL International is collaborating with the EBR Consortium so that XBRL enables the improvements in business reporting content over time.

4. XBRL Support Technologies for Decision Making

This section presents technologies that integrate with XBRL to create accounting systems which in turn, can be used for decision making. Current tools are been upgraded to support XBRL, from tools that just support for taxonomy and instance creation and validation, packages like

[7] http://xbrl.emporia.edu/
[8] http://www.musing.eu/
[9] http://www.xbrl-ntp.nl/
[10] http://xbrl.rienks.biz/
[11] http://www.ebr360.org/

Microsoft Office and ERP (Enterprise Resource Planning) tools such as SAP. In the next subsection we present the technologies that support or will support XBRL to its full capability.

4.1. Web services and XBRL

Web services are defined as a software application identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via internet-based protocols.

The reason why Web services are the natural technology for deploying XBRL accounting system include: interoperability among different technologies and programming languages, i.e., platform neutral and they can connect across heterogeneous networks using ubiquitous Web-based standards and heterogeneous applications; reusability of components, no installation and tight integration of software; accessibility as legacy assets and internal applications can exposed and accessible on the Web and furthermore, they could to be accessible on any device, anywhere, anytime. In the same way XBRL are open standards developed by a consortium of companies, Web services are also openly developed mainly by the World Wide Web Consortium (W3C)[12] and other industry standard organizations (Oasis[13], WS-I[14]). Also, as we have seen previously, XBRL is based on most of the technologies used by Web services such as XML and XML Schema.

The typical use of Web services in conjunction with XBRL for a company would be to retrieve financial reports in XBRL format automatically by financial analysis programs, eliminating the need to first locate and download the information (one of the properties of the definition of Web services is its ability to being discovered in transparent way).

Currently, there are several examples of Web tools for analyzing companies reports:

- The Danish Commerce and Companies Agency (DCCA)[15] has created a tool, called Digiregn web, for creating and analyzing documents in XBRL. It allows users to create, modify and validate documents according to selected taxonomies. Documents are presented to users as forms to edit and modify.
- The SEC is providing a drive test for a drive for a Web tool, Interactive Financial Report Viewer[16], for companies that have submitted their fillings in the currently voluntary program

It is worth noting that these tools are not still taking advantage of full Web services capabilities

4.2. XBRL taxonomies and Ontologies

There are several similarities between defining XBRL taxonomies and creating ontologies. Ontologies can be seen as part of the knowledge engineering domain. In fact, ontologies are engineered artifacts aimed at representing a shared, consensual conceptualization of the knowledge of a given domain [gruber, 95]. The use of *ontologies* [7] has been a recourse used in other fields in order to *integrate* the information, to *communicate* what people has achieved, to *adapt* the goals of the organization and to *support* the efficiency of the processes. Therefore, it means that ontologies help achieving some desirable qualities such as reusability providing formal representations, searchability providing meta-data as an index into information, reliability performing consistency checking, etc. Therefore, ontologies allow us to add semantics to data so that different software components can share information in a homogeneous way. Common uses of ontologies include communication between people and organizations and interoperability between

[12] http://www.w3.org/
[13] http://www.oasis-open.org/
[14] http://www.ws-i.org/

[15] http://www.eogs.dk/
[16] http://www.sec.gov/spotlight/xbrl/xbrlwebapp.htm

systems, i.e., translation of modelling methods, paradigms, languages and software tools.

Ontologies can be used a way to formalize concepts in financial domains to create better taxonomies, i.e., to model data, concepts, terms and relationships, processes and activities of the financial reporting supply chain. One advantage of using ontologies that there is a lot of work performed and tools such as Protégé that can be used in conjunction with XBRL technologies. The major benefict of using ontologies would be the transparent integration of different financial systems. An example could be the use of different names for the same concept in different departments of the same organization such as personnel and personnel expenses; instead of manually merging those accounts, a possibility is to access them through ontologies.

On the other hand, ontological engineering refers encompasses a set of activities conducted during conceptualization, design, implementation and deployment of ontologies [3]. Previous works in ontological engineering processes can be adapted to processes for XBRL engineering, i.e., processes for defining and extending taxonomies. In this respect, Piechocki et al [6] have defined a process model for engineering XBRL taxonomies based on software and ontology engineering. The problem is that the creation of ontologies is not straight forward and there are several modelling methodologies and guidelines to do so.

The use of ontologies is highly related to the semantic Web and make use of the technologies described with Web services in addition to further standards such as OWL (Ontology Web Language)[17] and the RDF (Resource Description Framework)[18]. As before, all these standards are open and developed by the W3C

5. Conclusions and Future Work

XBRL (eXtensible Business Reporting Language) is becoming the standard for reporting financial data since gained support from mayor regulators like SEC in the US and in Europe with the Basel II framework.

XBRL itself is just a language that enables preparation, analysis and communication of business information. Those benefits include cost savings, greater efficiency, accuracy and reliability to all activities involved in supplying or using financial data. The popularity of XBRL as an open standard is the great support achieved by XBRL International,a not-for-profit consortium consisting of about 450 members around the world by the major accounting firms, software vendors like Microsoft, IBM or SAP, stock exchanges, banks, data providers accounting bodies, etc.

This paper provided an overview of XBRL and related technologies including some extension such as the semantic Web and Web services as future challenges when implementing XBRL capable tools.

Future work will be to analyse and tackle problems that are still not solved with the implementation of XBRL technologies. For example, one problem that remains is that does not completely resolve the problem of diverse accounting practices in different countries. We briefly described how XBRL ontologies could be used to tackle this problem.

Acknowledgments

We thank the SpanishMinistry of Education and Science for supporting this research (TIN2004-06689-C03-01).

References

[1] Bergeron, B. Essentials of XBR: Financial Reporting in the 21st Century, John Wiley and Sons, NJ, 2003

[2] Devedžic, V., Understanding Ontological Engineering, in *Comm. of the ACM*, Vol. 45, No. 4, pp. 136-144, 2002

[3] Gruber, T., Towards Principles for the Design of Ontologies used for Knowledge Sharing. *International Journal of Human-Computer Studies*, Vol. 43, No. 5/6, pp. 907-928, 1995

[4] Hodge, F.D., Kennedy, J.J., Maines, L. Does Search-Facilitating Technology Improve the Transparency of Financial Reporting?, The Accounting Review, Vol 79, no. 3, pp. 687-703, 2004

[17] http://www.w3.org/TR/owl-features/

[18] http://www.w3.org/RDF/

[5] Lara R., Cantador I., Castells P., XBRL Taxonomies and OWL Ontologies for Investment Funds, 1st International Workshop on Ontologizing Industrial Standards at the 25th International Conference on Conceptual Modeling (ER2006), Tucson, Arizona, USA, November 6-9, 2006

[6] Piechocki M, Felden C, XBRL Taxonomy Engineering. Definition of XBRL Taxonomy Development Process Model. In Proceedings of the Fifteenth European Conference on Information Systems (Österle H, Fahy M, Feller J, Finnegan P, Murphy C (2005) Re-Engineering a Financial Information Supply Chain with XBRL: An Exploration of Co-Operative IOS Design and Development. In Proceedings of the 13th European Conference on Information Systems (Bartmann D, Rajola F, Kallinikos J, Avison D, Winter R, Ein-Dor P, Becker Jr, Bodendorf F, Weinhardt C eds.), 1175-1186, Regensburg, Germany, 2007

[7] Uschold M. and Grüninger M., Ontologies: Principles, Methods, and Applications, *Knowledge Engineering Review*, Vol. 11, No. 2, pp. 93—155, 1996

Software Engineering from an Engineering Perspective: SWEBOK as a Study Object

Alain Abran[a,b], Kenza Meridji[b], Javier Dolado[a]
a Universidad del País Vasco/Euskal Herriko Unibertsitatea
b Ecole de technologie supérieure - Université du Québec

Abstract

Software engineering, as a discipline, is not yet as mature as other engineering disciplines and it lacks criteria to assess, from an engineering perspective, the current content of its body of knowledge as embedded in the SWEBOK Guide. What is then the engineering knowledge that should be embedded within software engineering? Vincenti, in his book 'What engineers know and how they know it' has proposed a taxonomy of engineering knowledge types. To investigate software engineering from an engineering perspective, these Vincenti's categories of engineering knowledge are used to identify relevant engineering criteria and their presence in SWEBOK.

Keywords – Software Engineering, SWEBOK, ISO 19759, Vincenti, Engineering Knowledge Types

1. Introducción

"Engineering is a problem-solving activity…dealing mainly with practical problems" (Vincenti [6]).

1.1. Overview

Software engineering (SE) is defined by the IEEE as: "The application of a systematic, disciplined, quantitative approach to the development, operation and maintenance of software, the application of engineering to software' (IEEE 610.12) [3]. Of course, software engineering when compared to mechanical and electrical engineering is still an emerging engineering discipline not as mature as other classical engineering disciplines.

Much of the research work in software engineering has focused to date on developing methods, techniques and tools; much less research work has been carried out on:
exploring the engineering foundations of software engineering, including identifying the software engineering fundamental principles (FP), and next investigating on to apply them in research and practice.
Developing an international consensus on the software engineering body of knowledge and next ensuring a comprehensive coverage from an engineering perspective.

1.2. SE body of knowledge

Achieving consensus by the profession on a core body of knowledge is a key milestone in all disciplines, and has been identified by the IEEE Computer Society as crucial for the evolution of software engineering towards professional status. The Guide to the Software Engineering Body of Knowledge (SWEBOK Guide) [2,4]. The content of each knowledge area in the 2004 version of SWEBOK Guide was developed by domain experts and extensively reviewed by an international community of peers. This Delphi-type approach, while very extensive and paralleled by national reviews at the ISO level, did not specifically address the engineering perspective, nor did it provide a structured technique to ensure the completeness and full coverage of fundamental engineering topics. Therefore, it did not provide sufficient evidence that it had adequately tackled the identification and documentation of the knowledge expected to be present in an engineering discipline.

For the next update of the SWEBOK Guide research work has been initiated to analyze the content of the SWEBOK Guide in a structured

way in order to understand to which extent it does indeed include knowledge types typical of engineering disciplines, and to identify engineering knowledge could be missing. A challenge of course consists in figure out the criteria to be verified from an engineering perspective since, in the traditional engineering disciplines, such criteria have not been explicitly described in the generic engineering literature.

This paper presents an approach to identify engineering criteria to support such research needs. This paper is organized as follows: Section 2 introduces Vincenti's engineering viewpoint, and section 3 presents a set of models developed to facilitate the use of Vincenti's concepts for the analysis of an engineering discipline. Section 4 comments on a mapping of Vincenti's engineering design concept to the SWEBOK Guide software engineering design concept and on the Quality knowledge area. Section 5 presents a summary and future research directions.

2. Vincenti's Engineering Viewpoint

2.1. Overview and context

Vincenti, in his book [6], *What engineers know and how they know it*, proposed a taxonomy of engineering knowledge based on the historical analysis of five case studies in aeronautical engineering covering a roughly fifty-year period. He identified different types of engineering knowledge and classified them in six categories:

1 - Fundamental design concepts,
2 - Criteria and specifications,
3 - Theoretical tools,
4 - Quantitative data,
5 - Practical considerations, and
6 - Design instrumentalities.

Furthermore, Vincenti stated that this classification is not specific to the aeronautical engineering domain, but can be transferred to other engineering domains. However, he did not provide documented evidence of this applicability and generalization to other engineering disciplines, and no author could be identified as having attempted to do so either.

Vincenti provides a categorization of engineering design knowledge and the activities that generate it. However, the divisions are not entirely exclusive; some items of knowledge can contain the knowledge of more than one category. From Vincenti's definitions of each engineering knowledge-type category, a number of characteristics were identified; the goals of each category have also been identified, and these are listed in Table 1.

2.2. Related work

Maiebaum [5] was one of the first to identify the potential usefulness of Vincenti work for software engineering. However, since this classification of engineering knowledge had not been used to analyze other engineering disciplines, Abran & Meridji [1] modeled the embedded knowledge types descriptions for a partial analysis of the SWEBOK Guide. In particular, they investigated the engineering design concepts since at first glance there seemed to be a disconnect between the SWEBOK Guide design concept and Vincenti's description of engineering design. This is discussed in the next section.

3. Modeling of engineering knowledge

3.1. Overview

In [1] it was observed that Vincenti's categories are not mutually exclusive: it is therefore important to understand the relationships between them. Their initial modeling of Vincenti's categories of engineering knowledge is presented in Figure 1. This figure illustrates that, in seeking a design solution, designers move up and down within categories, as well as back and forth from one category to another.

Table 1. Vincenti: Engineering knowledge categories and goals

Engineering Knowledge Category	Goals
Fundamental design concepts	Designers embarking on any ***normal design*** bring with them ***fundamental concepts*** about the device in question.
Criteria and specification	To design a device embodying a given operational principle and normal configuration, the designer must have, at some point, ***specific requirements*** in terms of hardware.
Theoretical tools	To carry out their design function, engineers use a wide range of ***theoretical tools***. These include intellectual concepts as well as mathematical methods.
Quantitative data	Even with fundamental concepts and technical specifications at hand, mathematical tools are of little use without ***data*** for the ***physical properties*** or other ***quantities*** required in the formulas. Other kinds of data may also be needed to ***lay out details of the device*** or to ***specify manufacturing processes*** for production.
Practical considerations	To complement the theoretical tools and quantitative data, which are not sufficient. Designers also need ***less sharply defined considerations*** derived from experience.
Design instrumentalities	Besides the analytical tools, quantitative data and practical considerations required for their tasks, designers need to know ***how to carry out those tasks***. How to ***employ procedures*** productively constitutes an essential part of design knowledge.

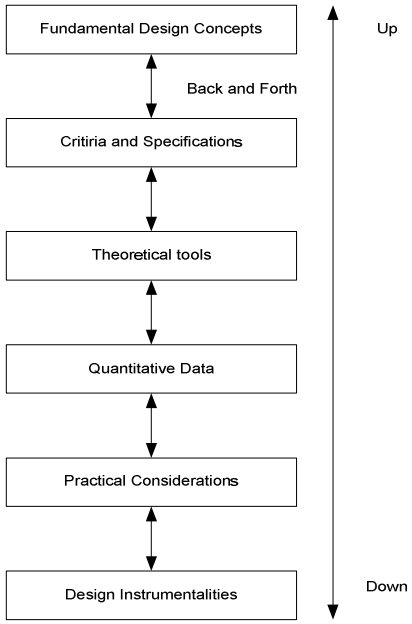

Figure 1. Vincenti's classification of engineering knowledge

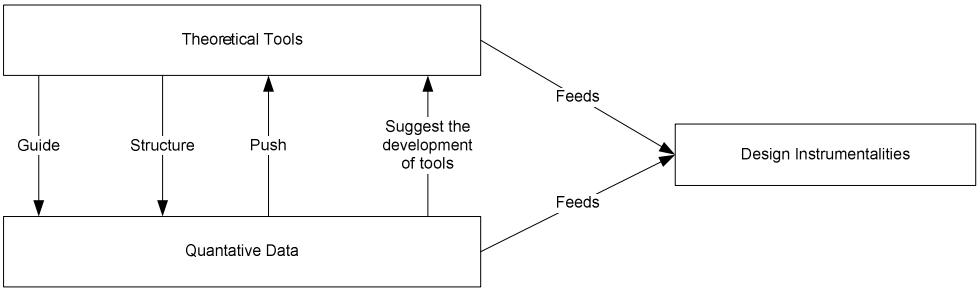

Figure 2: Relationships between theoretical tools & quantitative data

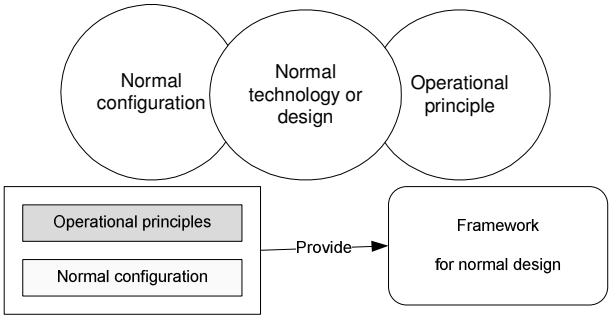

Figure 3: Relationships between normal configuration, operational principles & normal design

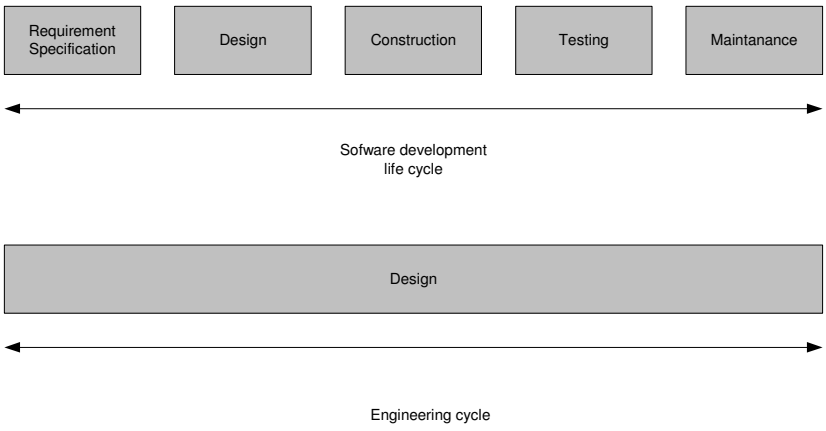

Figure 4. Design according to Vincenti vs. design in the software engineering life cycle

It was also noted that three categories (theoretical tools, quantitative data and design instrumentalities) are related in the following manner: theoretical tools guide and structure the data, while quantitative data suggest and push the development of tools for their presentation and application – see Figure 2. Furthermore, both theoretical tools and quantitative data serve as input for design instrumentalities, while appropriate theoretical tools and quantitative data are needed for technical specifications. This section presents some of their models to illustrate the relationships across these engineering concepts.

3.2. Fundamental design concepts

The goal of 'fundamental design concepts', according to Vincenti, is as follows: "*Designers setting out on any normal design bring with them fundamental concepts about the device in question,"* which means the definition of fundamental concepts related to the device by the designer. Fundamental design elements are composed of four elements; operational principles, normal configuration, normal technology and concepts in people's minds. At first, these concepts exist only in the designer's mind:

– **Operational principles** define the essential fundamental concept of a device. "How its characteristic parts… fulfill their special functions in combination to [sic] an overall operation which archives the purpose." The operational principles must be known by the designers first and constitute the basic components for the design, whereas operational principles are abstract, and the design moves from abstract concepts to precise concepts.

– **Normal configuration** is "the general shape and arrangement that are commonly agreed to best embody the operational principle."

– Normal technology is "*the improvement of the accepted tradition or its application under new or more stringent conditions.*" Design, in Vincenti, "denotes both the content of a set of plans (as in the design for a new airplane) and the process by which those plans are produced." There are two types of design: **normal design** and radical design. The latter is a kind of design that is unknown to the designer, and where the designer is not familiar with the device itself. The designer does not know how the device should be arranged, or even how it works. The former is a traditional design, where the designer knows how the device works. The designer also knows the traditional features of the device. This type of design is also the design involved in normal technology, which was mentioned earlier. In conclusion, "*normal design is evolutionary rather than revolutionary.*" Finally, a normal configuration and operational principles together provide a framework for normal design – Figure 3. In Vincenti, a normal technology, or design, is part of a normal configuration and of a related operational principle.

3.3. Criteria and specifications

The goal for 'criteria and specifications' can be expressed as follows: "*To design a device embodying a given operational principle and normal configuration, the designer must have, at some point, specific requirements in terms of hardware.*" The designer designs a device meeting specific requirements which include a given operational principle as well as a normal configuration. At first, the design problem must be well defined. Then, the designer translates general quantitative goals into specific quantitative goals: the designer assigns values or limits to the characteristics of the device which are crucial for engineering design. This allows the designer to provide the details and dimensions of the device that will be given to the builder. Furthermore, the output at the problem definition level is used, in turn, as input to the remaining design activities that follow. These specifications are more important where safety is involved, as in the case of aeronautical devices. The criteria on which the specifications are based become part of the accumulating body of knowledge about how things are done in engineering.

3.4. Theoretical tools

Theoretical tools are used by engineers to carry out their design. The goal of the 'theoretical tools' category is expressed by Vincenti as follows: "*To carry out their design function, engineers use a wide range of **theoretical tools**. These include intellectual concepts as well as mathematical*

methods". Intellectual concepts (such as design concepts, mathematical methods and theories) are tools for making design calculations. Both design concepts and methods are part of science.

In the first class of theoretical tools are mathematical methods and theories composed of formulas, either simple or complex, which are useful for quantitative analysis and design. This scientific knowledge must be reformulated to make it applicable to engineering. The engineering activity requires that thoughts be conceived in people's minds.

In the second class of theoretical tools are intellectual concepts, which represent the language expressing those thoughts in people's minds. They are employed first in the quantitative conceptualization and reasoning that engineers have to perform before they carry out the quantitative analysis and design calculations, and then again while they are carrying them out.

3.5. Quantitative data

The goal of 'quantitative data' is to lay down "the **physical properties** or other **quantities** required in the formulas. Other kinds of data may also be needed to **lay out details of the device** or to **specify manufacturing processes** for production." Besides fundamental concepts and technical specifications, the designers also need quantitative data to lay out details of the device. These data can be obtained empirically, or in some cases they can be obtained theoretically. They can be represented in tables or graphs.

These data are divided into two types of knowledge: prescriptive and descriptive.

- Descriptive knowledge is "*knowledge of how things are*." It includes physical constants, properties of substances and physical processes. In some situations, it refers to operational conditions in the physical world. Descriptive data can also include measurement of performance.
- Prescriptive knowledge is "*knowledge of how things should be to attain a desired end.*" An example might be: "In order to accomplish this or organize this, arrange things this way."

Operational principles, normal configuration and technical specifications are prescriptive knowledge, because they prescribe how a device should satisfy its objective.

3.6. Practical considerations

According to Vincenti, the goal of 'practical considerations' is "to complement the role of theoretical tools and quantitative data which are not sufficient. Designers also need for their work **less sharply defined considerations** derived from experience." This kind of knowledge is prescriptive in the way that it shows the designers how to proceed with the design to achieve it. Vincenti refers to practical considerations as constituting non codifiable knowledge derived from experience, unlike theoretical tools and quantitative data which are very precise and codifiable because these are derived from intentional research. This category of engineering knowledge is needed by designers as a complement to theoretical tools and quantitative data. These practical considerations are learned on the job, rather than at school or from books. They are not to be formalized or programmed. They are derived from design, as well as from production and operation. The practical consideration derived from production is not easy to define and cannot be codified, and a prototype is highly recommended to check the designer's work. An example of a practical consideration from operation is the judgment that comes from the feedback resulting from use.

3.7. Design instrumentalities

The goal of 'design instrumentalities' in the engineering design process required for the engineer's tasks is "*to know **how to carry out those tasks**. How to **employ procedure** productively constitutes an essential part of design knowledge.*" Having the analytical tools, quantitative data and practical considerations at hand, designers also need procedural knowledge to carry out their tasks, as well as to know how to employ these procedures.

Design instrumentalities contain instrumentalities of the process, the procedures, judgment and ways of thinking. The latter are less tangible than procedures and more tangible than judgment; an example of ways of thinking is 'thinking by

analogy'. Judgment is needed to seek out design solutions and make design decisions.

4. Analysis of the SWEBOK using engineering knowledge types concepts

4.1. The engineering design process in Vincenti

According to Vincenti, the engineering "design" concept "denotes both the content of a set of plans (as in the design for a new airplane) and the process by which those plans are produced." In Vincenti's view, design is an iterative and complex process which consists of plans for the production of a single entity, such as an airplane (device), how these plans are produced, and, finally, the release of these plans for production.

Vincenti mentions that there are two types of design in engineering, **normal** and **radical**. In the former, the designer knows how the device works, how it should be arranged and what its features are. In the latter, the device is new to the engineer who is encountering it for the first time. Therefore, the engineer does not know how it works or how it should be organized.

He also mentions that design is a multilevel and hierarchical process. The designer starts by taking the problem as input. The design hierarchies start from the project definition level, located at the upper level of the hierarchy where problems are abstracted and unstructured. At the overall design level, the layout and the proportions of the device are set to meet the project definition. At level 3, the project is divided into its major components. At level 4, each component is subdivided. At level 5, the subcomponents from level 4 are further divided into specific problems. At the lower levels, problems are well defined and structured. The design process is iterative, both up and down and horizontally throughout the hierarchy.

4.2. The engineering process and the Design concept in the SWEBOK Guide

The SWEBOK Guide is composed of ten knowledge areas, each represented by one chapter in the SWEBOK Guide. The Software Requirements KA (KA) is composed of four phases of software requirements: elicitation, analysis, specification and validation. The elicitation phase is the process of deriving requirements through observation of existing systems. Requirements specification is the activity of transforming the requirements gathered during the analysis activity into a precise set of requirements. Software Requirements Specifications describe the software system to be delivered. In the requirements validation phase, the requirements are checked for realism, consistency and completeness.

Software design is defined in [2,4] as both "the process of defining the architecture, components, interfaces, and other characteristics of a system or component" and "the result of [that] process." Software ***design*** in the software engineering life cycle is an activity in which software requirements are taken as input to the software design phase for analysis. "*Software requirements express the needs and constraints placed on a software product that contribute to the solution of some real-world problem."*

The result will be the description of the software architecture, its decomposition into different components and the description of the interfaces between those components. Also described will be the internal structure of each component and the related program.

4.3. Design KA: mapping between Vincenti and the SWEBOK Guide

The analysis of the term 'design' in both Vincenti and the SWEBOK Guide is presented in Table 4: it can be observed that it is defined significantly differently in the two documents, that is, design in engineering according to Vincenti is not limited to design as described in the SWEBOK Guide: in Vincenti, it goes far beyond the scope of the SWEBOK, that is: it is composed of the whole of the software engineering life cycle, as illustrated in Figure 8, whereas all the activities of software life cycle, like the requirements phase, the design phase, the construction phase and the testing phase) map to a single phase in the engineering cycle, that is, design. These activities do not necessarily take place in the same order: for instance, testing in engineering starts right at the beginning, at the problem definition level, and goes on until the final release of the plans for production, while in the software engineering life

cycle, as defined generically in the SWEBOK Guide, testing starts after the construction phase; on the other hand, the set of V&V concepts are spread out throughout the lifecycle in SWEBOK.

The detailed mapping between the different design levels in engineering and in the software engineering life cycle is presented in Table 2.

4.4. Identification of engineering concepts in the SWEBOK Software Quality KA

An analysis of the engineering content within the SWEBOK Guide using one of its ten KAs as a case study, that is, Software Quality is presented [1]. This analysis is based on the models of engineering knowledge described earlier. These models give us a very descriptive analysis of the various key elements contained in each of the corresponding engineering knowledge areas. This allows to make an appropriate mapping between the different categories of the engineering knowledge area and software quality. It helps in identifying the engineering elements contained in this topic, as well as the missing ones. As a result, it looks into the software quality area from an engineering perspective. Table 3 describes the mapping between the corresponding characteristics for the classification of engineering knowledge and the related software quality topics. This analysis can provide useful insights into possible strengths and weaknesses of the software quality topic: it helps categorize the knowledge contained in the Software Quality KA of the SWEBOK Guide: for instance, it covers all categories of engineering knowledge from an engineering viewpoint, but this does not mean that it is complete and inclusive.

5. Summary

Software engineering, as a discipline, is certainly not yet as mature as other engineering disciplines and it lacks well recognized fundamental principles, as well as criteria to assess, from an engineering perspective, the proposals put forward as statement of fundamental principles as well as the current content of its body of knowledge as embedded in the SWEBOK Guide. In this paper we have looked into an approach to identified engineering criteria that should be embedded within software engineering. In particular, we have looked at Vincenti at the taxonomy of engineering knowledge types proposed by Vincenti.

In particular, various models of the characteristics of the Vincenti's knowledge type have been illustrated. Next these concepts have been used to gain some insights into the some of the engineering concepts currently present and documented in the quality knowledge area of the SWEBOK Guide, but however labeled differently. The work presented here has involved investigating this engineering perspective, first by analyzing the Vincenti classification of engineering knowledge, and second by comparing the design concept in Vincenti vs. the design concept in the SWEBOK Guide.

The result of this analysis was to show that the design issue in Vincenti is not limited to the design issue in the SWEBOK Guide: Design in engineering according to Vincenti is not limited to design as described in the SWEBOK Guide: it goes beyond that, in that it is composed of the whole of the software engineering life cycle.

Finally, the SWEBOK Software Quality KA was selected as a case study and analyzed using the Vincenti classification as a tool to analyze this KA from an engineering perspective. This analysis was carried out to identify some of the strengths and weaknesses of the breakdown of topics for the Software Quality KA. It has shown that all the categories of engineering knowledge described by Vincenti are present in this KA of the SWEBOK; that is, it addresses the full coverage of all related engineering-type knowledge. This does not mean, however, that it is all-inclusive and complete, but only that the coverage extends to all categories of engineering knowledge from an engineering viewpoint.

The next stage of this R&D project will focus on investigating the application of Vincenti's engineering knowledge to the analysis of proposed software engineering principles.

Acknowledgements

This research project has been funded partially by the European Community's Sixth Framework Programme – Marie Curie International Incoming Fellowship under contract MIF1-CT-2006-039212.

Table 2. Mapping of the design process in engineering vs. the software engineering life cycle

Levels	Description of the design process in Vincenti engineering perspective	Corresponding set of concepts in SWEBOK
1	Project Definition	Requirements
2	Overall design – component layout of the airplane to meet the project definition.	Specification
3	Major component design – division of project into wing design, fuselage design, landing gear design, electrical system design, etc.	Architecture of the system
4	Subdivision of areas of component design from level 3 according to the engineering discipline required (e.g. aerodynamic wing design, structural wing design, mechanical wing design)	Detailed design
5	Further division of the level 4 categories into highly specific problems	Construction

Table 3: Quality concepts in the SWEBOK Guide using Vincenti's classification

Engineering Knowledge Category	Corresponding Characteristics	SWEBOK – quality related concepts
Fundamental design concepts	• About the design • Designers must know the operational principle of the device • How the device works • Normal configuration • Normal design • Other features may be (opened?)	• Planning the software quality process • Quality characteristics of the software (QI), (QE), (QIU) • Software quality models • Quality assurance process • Verification process • Validation process • Review process • Audit process
Criteria and specification	• Specific requirement of an operational principle • General qualitative goals • Specific quantitative goals laid out in concrete technical terms • The design problem must be "well defined". • Unknown or partially understood criteria • Assignment of values to appropriate criteria • This task takes place at the project definition level.	• Quality objective to be specified • Characteristics of quality tools • Software characteristics • Criteria for assessing the characteristics
Theoretical Tools	• Mathematical methods and theories for making design calculation • Intellectual concepts for thinking about design • Precise and codifiable	• Verification process model • Formal methods • Testing • Theory measurement • Verification/proving properties • TQM (Total Quality Management)

Quantitative data	• Specify manufacturing process for production • Display the detail for the device • Data essential for design • Obtained empirically • Calculated theoretically • Represented in tables or graphs • Descriptive knowledge • Prescriptive knowledge • Precise and codifiable	• Quality measurement • Experimental data • Empirical study • E.g. the process of requirement inspection • Value and cost of quality
Practical Considera-tions	• Theoretical tools and quantitative data are not sufficient. Designers also need considerations derived from experience. • It is are difficult to find them documented. • They are also derived from production & operation. • This knowledge is difficult to define. • Its defies codification • The practical consideration derived from operation is judgment. • Rules of thumb.	• Application quality requirements • Defect characterization
Design Instrumenta-lities	• Knowing how • Procedural knowledge • Ways of thinking • Judgment skills	• Quality assurance procedures • Quality verification procedures • Quality validation procedures • SQM process tasks & techniques • Management techniques • Measurement techniques • Project planning and tracking • Quality assurance process • Verification process • Validation process • Review process • Audit process

References

[1] Abran, A., Meridji, K., 'Analysis of Software Engineering from An Engineering Perspective', European Journal for the Informatics Professional ,vol. 7, No. 1, February , 2006 , pp. 46-52 . www.upgrade-cepis.org Upgrade: ISSN 1684-5285 Novática: ISSN 0211-2124

[2] Abran, A., Moore, J., Bourque, P., Dupuis, R., Tripp, L. (2005), Guide to the Software Engineering Body of Knowledge – SWEBOK, IEEE Computer Society Press, Los Alamitos, URL: http://www.swebok.org

[3] IEEE 610.12-1990 (1990), IEEE Standard Glossary of Software Engineering Terminology, Institute of Electrical and Electronics Engineers ISBN: 155937067X. 84 pages.

[4] ISO/IEC TR 19759-2005 (2005), Guide to the Software Engineering Body of Knowledge (SWEBOK), International Organization for Standardization - ISO, Geneva, 2005

[5] Maieubaum, T., 'Mathematical foundations of software engineering: a roadmap', 22nd International Conference on The future of Software Engineering, June 4 - 11, 2000, Limerick Ireland, Pages 161-172.

[6] Vincenti, W. G. (1990). *What engineers know and how they know it*. Baltimore, London: The Johns Hopkins University Press.

La Gestión de Equipos para la Mejora del Proceso Software

Esperança Amengual, Antònia Mas
Departament de Matemàtiques i Informàtica
Universitat de les Illes Balears
07122 Palma de Mallorca - Illes Balears, Spain
{eamengual, antonia.mas}@uib.es

Resumen

En las organizaciones modernas el trabajo en equipo se considera un factor clave para el éxito empresarial. El interés por la cultura de equipos ha culminado con la presencia de numerosos artículos que analizan los diferentes aspectos para mejorar las prácticas de trabajo en equipo. Debido a que los proyectos software se desarrollan normalmente en equipos, la mejora del trabajo en equipo en las empresas de desarrollo de software debería también ser considerado como un aspecto importante. En estas organizaciones, los programas de mejora del proceso software basados en los estándares de madurez internacionales son un tema actual en la investigación dentro del campo de la ingeniería del software. En este artículo, en primer lugar se establecen los factores clave del trabajo en equipo para tener éxito en los proyectos de desarrollo de software. En segundo lugar, se analizan estos factores en el marco de la ISO/IEC 15504 como modelo de referencia para la mejora del proceso software

Palabras clave : Trabajo en equipo, ISO/IEC 15504, Gestión de la calidad del Software.

1. Introducción

Hoy en día, el trabajo en equipo se ha popularizado como solución al objetivo principal de un gran número de compañías: producir al menor coste. Si consideramos a los empleados como el recurso más importante que tiene una organización, el trabajo en equipo se revela como la manera más eficiente de conseguir este objetivo. Las organizaciones modernas tienen unas expectativas sobre sus empleados que van más allá de la simple realización del trabajo para contribuir al éxito empresarial. Tal y como se demuestra en numerosos artículos, existe un interés creciente en la cultura de equipos que se refiere a la habilidad de trabajar con éxito en un equipo [1]. Sin embargo, aunque la mayoría de organizaciones consideran la facilidad de trabajar en equipo como una habilidad importante a la hora de seleccionar a sus empleados, todavía queda mucho trabajo por hacer para conseguir una cultura de trabajo en equipo real.

La naturaleza es sabia y nos proporciona modelos de trabajo en equipo, como las comunidades de abejas o de hormigas, en las que el objetivo final se consigue uniendo los esfuerzos individuales. Estas organizaciones naturales son buenos ejemplos a seguir que demuestran que la interdependencia entre los miembros del equipo es una característica clave de los equipos con éxito [2]. Sin embargo, esta predisposición natural hacia el trabajo en equipo parece no ser tan evidente en el caso de los seres humanos. El trabajo en equipo es un estilo de trabajo que no todas las personas están dispuestas a aceptar. A veces, el espíritu de trabajo individualista puede ser un obstáculo importante a vencer. Uno de los mayores problemas en las organizaciones es reunir a un grupo de personas para cumplir un objetivo empresarial, ya que todas ellas tienen necesidades, intereses, conocimiento, experiencias, expectativas y motivaciones diferentes.

Aunque no debemos considerar al trabajo en equipo como la panacea, la investigación en la formación y el rendimiento de grupos de trabajo ha sido el centro de atención de diferentes especialistas durante las dos últimas décadas. Este hecho se puede justificar considerando el papel tan importante que los equipos pueden tener llevando a cabo tareas efectivas en la organización. De acuerdo con el Dr. Charles J. Margerison, que afirma que “dependemos de las competencias y de la efectividad de los equipos”, se ha escrito mucho acerca de las competencias individuales en el trabajo en comparación con las

competencias de equipo que han recibido poca atención. Las competencias individuales son importantes, pero necesitan ser estudiadas en el contexto de lo que un equipo necesita para funcionar bien [3].

2. Equipos en proyectos software

En las empresas de desarrollo de software, la gran demanda de nuevos sistemas junto con el incremento de la complejidad de los mismos, hacen que el proceso de desarrollo de software se considere una actividad de equipo. Por tanto, en estas organizaciones en particular, el interés por el trabajo en equipo no debería ser una excepción.

Algunos trabajos demuestran que este tema no ha pasado desapercibido. En [4] se considera que la coordinación y la comunicación en un equipo software son aspectos clave que deben tenerse en cuenta. Del mismo modo, la interacción social también es considerada un punto importante para el éxito en los proyectos software. En [5], su autor, siguiendo la misma línea que otros artículos también considerados en este trabajo, destaca que "al parecer los aspectos humanos del desarrollo del software son más importantes que los aspectos tecnológicos para un mejor rendimiento". Para analizar esta afirmación el artículo antes mencionado presenta una investigación en donde se analizan los efectos de la personalidad en la productividad de un equipo. En particular, el estudio busca determinar el efecto de la personalidad del líder del proyecto, así como el efecto de las personalidades de los miembros del equipo en los resultados finales.

Considerando también los aspectos humanos como un factor clave a controlar en un equipo de desarrollo de software, es posible encontrar otras publicaciones. En [6] se demuestra que los roles de equipo descritos por R. Meredith Belbin [7] son útiles para mejorar la efectividad de los equipos de desarrollo de software. En este artículo se utiliza el cuestionario de Belbin como instrumento de recogida de datos individuales para analizar tres equipos de desarrollo de software que trabajan en entornos diferentes. En [8] sus autores presentan un experimento para demostrar la utilidad de formar equipos basados en los roles de equipo de Belbin y llegan a la conclusión de que el cuestionario de Belbin resulta útil para identificar características que los miembros del equipo deben poseer para que funcione mejor.

En [9] su autora señala que la mayoría de problemas en proyectos software "son debidos a problemas con las personas, no a problemas técnicos". Aunque producir software de calidad es una actividad técnica, el software lo producen personas. Se han propuesto diferentes modelos de madurez y modelos de procesos, pero los problemas todavía continúan.

Las diferentes investigaciones que se mencionan en este apartado afirman de una manera u otra que es necesario considerar aspectos específicos de trabajo en equipo para tener éxito en un proyecto de desarrollo de software.

3. El trabajo en equipo en los Modelos de Madurez

El Software Engineering Institute (SEISM), después de desarrollar el Capability Maturity Model como un modelo descriptivo de las características de una organización a un nivel particular de madurez del proceso software [10], desarrolló el Team Software ProcessSM (TSPSM), un modelo prescriptivo para equipos de desarrollo de software. Tal y como se define en el informe técnico del SEI que relaciona TSP con CMM [11], "TSP es un proceso de alta madurez para equipos de proyectos. Contiene un conjunto de procesos, procedimientos, guías y herramientas adaptables a los equipos de proyectos para que puedan se utilizados en la producción de software de gran calidad, a tiempo y dentro del presupuesto establecido". En [12] se estipulan algunos de los resultados de proyectos que han adoptado TSP. Estos resultados muestran que los equipos TSP desarrollan un software libre de defectos y cumplen los plazos de entrega al mismo tiempo que mejoran la productividad.

En [13] se examina la relación entre el Capability Maturity Model y el Team Software Process como tecnologías complementarias y se analiza el grado con el que CMM es dirigido por TSP. Otros artículos publicados [14, 15], muestran la utilidad de TSP para alcanzar un determinado nivel de madurez según CMM.

Gracias a la experiencia particular en la mejora de desarrollo de software en ocho pequeñas y medianas empresas de nuestro entorno [16, 17],

las autoras de este artículo estamos de acuerdo en que la mejora del trabajo en equipo debería ser considerada como un factor clave para mejorar los procesos de desarrollo de software. Siguiendo con nuestra investigación en la aplicabilidad del estándar internacional ISO/IEC 15504 en pequeñas y medianas empresas de desarrollo de software [18], en este artículo se analiza la medida en que este estándar considera los aspectos de trabajo en equipo. Para ello, en primer lugar se establecen los factores de trabajo en equipo que se consideran clave para tener éxito en un proyecto software. En segundo lugar, a partir de un análisis exhaustivo del estándar, se determina el grado con el que estos factores son considerados explícita o implícitamente por dicho estándar.

4. Factores clave del trabajo en equipo

En una iniciativa de mejora de procesos software, aunque la mayoría de proyectos son llevados a cabo por equipos de profesionales, los aspectos tecnológicos son los que suelen recibir más atención en comparación con la atención que se presta a la dinámica de equipos. A partir de nuestra investigación sobre el trabajo en equipo y, más concretamente, sobre el trabajo en equipos de desarrollo de software, y desde nuestra propia experiencia, en este apartado se establecen las características que todo equipo de desarrollo de software debería poseer.

4.1. Gestión del equipo

Tal y como se muestra en [19], cuando las personas trabajamos en grupo, existen dos aspectos separados que se deben tener en cuenta. El primero se refiere a las tareas que implica hacer el trabajo. Frecuentemente este es el único aspecto que el grupo considera. El segundo, es el proceso de trabajar en grupo: los mecanismos por los que el grupo actúa como una unidad.

De acuerdo con esta afirmación, pensamos que los equipos deben considerarse un recurso valioso que necesita ser gestionado al igual que se hace con los demás recursos en un proyecto. Así pues, para la correcta ejecución del equipo, se considera importante:

- La planificación. Definición de objetivos y tareas.
- La monitorización. Controlar que los objetivos se han cumplido de acuerdo con la planificación establecida.

Estos dos aspectos, básicos en toda gestión, deberían considerarse en particular para cada uno de los miembros del equipo, así como para el equipo como una entidad.

4.2. Coordinación

De acuerdo con [4], la coordinación y la eficiencia son necesarias para el éxito del trabajo en equipo. Estos dos aspectos del trabajo en equipo pueden deducirse de la propia definición de trabajo en equipo como “el trabajo de un equipo con referencia a la coordinación de esfuerzos y a la eficiencia colectiva”. Es de suponer que la eficiencia es el resultado esperado del esfuerzo de coordinación. Por tanto es relevante considerar como se puede conseguir esta coordinación.

El éxito en la coordinación requiere que cada miembro del equipo entienda:

- Quien es el responsable de cada parte del proyecto.
- Las relaciones entre las tareas asignadas a los diferentes miembros del equipo.
- Como progresa el trabajo con respecto a la planificación establecida.

4.3. Comunicación efectiva

La comunicación es probablemente el componente más esencial del trabajo en equipo [4]. De acuerdo con [2], en donde se identifica la comunicación inefectiva como uno de los seis factores que resultan negativos para el trabajo en equipo, es necesario definir un mecanismo de comunicación explícito que puede llevarse a cabo de distintas formas:

- Reuniones de proyecto (semanales, bisemanales o cualquier otra planificación que resulte apropiada)
- Los miembros del equipo pueden distribuir informes de progreso resumiendo el estado del proyecto en

cuanto a tiempos, expectativas y otros aspectos relacionados.
- Comunicación informal para mantener a los miembros del equipo interconectados.

4.4. Composición del equipo

Varios estudios sobre la mejora de la efectividad de los equipos de trabajo se basan en la composición del equipo como factor clave que puede afectar al rendimiento del proyecto. Una de las mayores contribuciones al análisis del rendimiento de los equipos en las organizaciones es la identificación de los roles de equipo. Un trabajo relevante en esta área es la teoría de roles de Belbin en la que su autor identifica ocho roles. En [20] se describe un modelo de equipo destinado a mejorar el rendimiento en un equipo de desarrollo de software. Este modelo describe una propuesta de estructura de las personas y de sus actividades para conseguir el éxito del proyecto. Se basa en seis objetivos clave que conducen al equipo y a los roles asociados.

Entonces para construir un buen equipo es deseable:

- Identificar los roles para llevar a cabo las diferentes tareas.
- Determinar las personas más convenientes para cada rol.
- Asignar responsabilidades.

4.5. Motivación

En las pequeñas y medianas empresas de desarrollo de software los equipos de trabajo normalmente están formados por pequeños grupos en los que el factor humano resulta crucial para el éxito. En [21] la motivación se considera un aspecto esencial para la efectividad del equipo. Aunque no resulta sencillo, es necesario mantener el entusiasmo y el acuerdo entre los miembros del equipo.

La motivación en un equipo puede conseguirse considerando los siguientes motivadores para cada miembro del equipo:

- Responsabilidad
- Interés en las tareas asignadas
- Cumplimiento de los objetivos
- Reconocimiento del trabajo realizado

5. ISO/IEC 15504. Aspectos de trabajo en equipo

ISO/IEC 15504 es un estándar internacional que resulta apropiado para cualquier organización, independientemente de su dominio o tamaño, que proporciona un enfoque estructurado para la evaluación de los procesos con el objetivo de conocer el estado de estos procesos para poder mejorarlos.

La parte 5 del estándar, *An exemplar process assessment mode*l basade en la ISO/IEC 12207 Amd 1 & 2 [22], publicada en Marzo de 2006, es una parte informativa que proporciona un ejemplo de un Modelo de Evaluación de Procesos que cumple con los requisitos de la parte normativa, ISO/IEC 15504-2, *Performing an assessment* [23].

La medida de la capacidad de un proceso, tal y como se define en la parte 2 del estándar, se basa en nueve atributos de proceso. Estos atributos se utilizan para determinar si un proceso ha alcanzado un nivel de capacidad determinado. Cada atributo mide un aspecto particular de la capacidad del proceso. Así, para medir el grado de cumplimiento de estos atributos, el estándar considera diferentes prácticas genéricas para cada uno de ellos.

Después de una revisión detallada de todas las prácticas genéricas en cada uno de los seis niveles que el estándar define, a continuación se analizan los factores clave del trabajo en equipo detallados en el apartado anterior para ver si son considerados dentro de estas prácticas genéricas.

5.1. Gestión del equipo

Los aspectos de planificación, más concretamente, la identificación de los recursos humanos y la definición de responsabilidades, son considerados en las prácticas genéricas que el estándar propone para medir el nivel 2 que se refiere a un proceso implementado de manera gestionada.

El nivel 3, *Established process*, asegura que el proceso antes gestionado se implementa siguiendo un proceso definido. En este nivel se considera explícitamente la asignación de recursos humanos para respaldar la ejecución del proceso.

En el nivel 5, *Optimizing process*, se consideran aspectos de compromiso, tanto a nivel

organización como a nivel del responsable del proceso.

Además, el estándar también destaca los factores humanos que influyen en la efectividad y en el despliegue total de los cambios del proceso acordados. En particular, considera el compromiso, la cultura organizacional y el riesgo como factores de gestión importantes.

La Tabla 1 muestra un resumen de los aspectos de gestión del equipo que el estándar considera en los diferentes niveles de capacidad.

Prácticas genéricas ISO/IEC 15504	Gestión del Equipo
Evaluación del Nivel 2	
Define responsibilities and authorities for performing the process	Definir responsabilidades
Identify and make available resources to perform de process according to plan	Identificar los recursos humanos y de infraestructura
Evaluación del Nivel 3	
Provide resources and information to support the performance of the defined process	Disponer, localizar y usar los recursos humanos requeridos
Evaluación del Nivel 5	
Define an implementation strategy based on long-term improvement vision and objectives	La gestión de la organización y los responsables de los procesos demuestran un compromiso con la mejora
Manage the implementation of agreed changes to selected areas of the defined and standard process according to the implementation strategy	Identificar y gestionar: el compromiso, la cultura organizacional y el riesgo

Tabla 1- Factor de Gestión del equipo

5.2. Coordinación

Aunque los aspectos de coordinación no se pueden observar directamente en las prácticas propuestas por el estándar, es posible deducir que algunos aspectos de coordinación son considerados en determinados niveles tal y como se muestra en la Tabla 2.

Prácticas genéricas ISO/IEC 15504	Coordinación
Evaluación del Nivel 2	
Manage the interfaces between involved parties	Gestionar las interfaces entre todas las partes involucradas

Evaluación del Nivel 4	
Establish quantitative objectives for the performance of the defined process, according to the alignment of the process with the business goals	Verificar los objetivos de cumplimiento del proceso con los responsables del proceso
Collect product and process measurement results through performing the define process	Informar de los resultados de las mediciones a los responsables de monitorizar el cumplimiento de los objetivos
Evaluación del Nivel 5	
Manage the implementation of agreed changes to selected areas of the defined and standard process according to the implementation strategy	Identificar y gestionar: conflicto / cohesión en los objetivos acordados

Tabla 2- Factor de Coordinación

5.3. Comunicación

De acuerdo con el estándar ISO/IEC 15504 es necesario establecer mecanismos formales de comunicación para asegurar que todos los participantes en un proceso puedan obtener la información necesaria para llevarlo a cabo. La Tabla 3 demuestra que la comunicación es un aspecto que se considera al completo en los diferentes niveles de capacidad.

Prácticas genéricas ISO/IEC 15504	**Comunicación**
Evaluación del Nivel 2	
Define responsibilities and authorities for performing the process	Comunicar responsabilidades
Identify and make available resources to perform de process according to plan	Disponer de la información necesaria para llevar a cabo el proceso
Manage the interfaces between involved parties	Asegurar la comunicación entre las partes involucradas
Evaluación del Nivel 3	
Provide resources and information to support the performance of the defined process	Disponer, localizar y usar la información requerida para llevar a cabo el proceso
Evaluación del Nivel 4	
Analyse process and product measurement results to identify variations in process performance	Proporcionar los resultados a los responsables de llevar a cabo el proceso

Evaluación del Nivel 5	
Manage the implementation of agreed changes to selected areas of the defined and standard process according to the implementation strategy	Comunicar los cambios del proceso a todas las partes afectadas
Evaluate the effectiveness of process change on the basis of actual performance against process performance and capability objectives and business goals	Disponer de un mecanismo para documentar e informar de los resultados de los análisis

Tabla 3- Factor de Comunicación

5.4. Composición

Los diferentes aspectos de la composición del equipo, como la identificación de los roles, la determinación de los participantes y la asignación de responsabilidades, están presentes en todos los niveles de capacidad considerados por el estándar (Tabla 4).

Prácticas genéricas ISO/IEC 15504	**Composición**
Evaluación de Nivel 2	
Define responsibilities and authorities for performing the process	Asignar responsabilidades
Manage the interfaces between involved parties	Asignar responsabilidades. Determinar las personas y los grupos involucrados en el proceso
Evaluación de Nivel 3	
Identify the roles and competencies for performing the standard process	Identificar los roles de ejecución del proceso. Identificar las competencias para llevar a cabo el proceso
Assign and communicate roles, responsibilities and authorities for performing the defined process	Asignar y comunicar los roles y las autoridades para llevar a cabo el proceso definido
Ensure necessary competencies for performing the defined process	Identificar las competencias apropiadas para el personal asignado
Identify process information needs in relation with business goals	Identificar los responsables del proceso
Define responsibilities and establish infrastructure to collect product and process measures	Definir responsabilidades para la recopilación de los datos
Evaluación de Nivel 4	
Identify process information needs in relation with business goals	Identificar los responsables del proceso

Define responsibilities and establish infrastructure to collect product and process measures	Definir responsabilidades para la recopilación de los datos
Evaluación de Nivel 5	
Manage the implementation of agreed changes to selected areas of the defined and standard process according to the implementation strategy	Identificar y gestionar: las habilidades, el liderazgo, el conocimiento, las capacidades

Tabla 3- Factor de Composición

5.5. Motivación

Este factor clave es un aspecto que el estándar tan solo considera en el nivel de capacidad 5, en donde se miden la satisfacción, la motivación y la moral de los empleados como indicadores de un proceso que se encuentra en el nivel de capacidad más alto (Tabla 5).

Prácticas genéricas ISO/IEC 15504	Motivación
Evaluación del Nivel 5	
Manage the implementation of agreed changes to selected areas of the defined and standard process according to the implementation strategy	Identificar y gestionar: satisfacción, motivación y moral

Tabla 3- Factor de Motivación

6. Conclusiones y trabajo futuro

En este artículo se han presentado los primeros resultados de una investigación que tiene por objetivo considerar los aspectos de trabajo en equipo como esenciales para tener éxito en los programas de mejora del proceso software.

En primer lugar, se han establecido los factores claves para el trabajo en equipo que se consideran importantes para un equipo de desarrollo de software. Así pues, considerando estos factores clave dentro del marco del estándar ISO/IEC 15504, se ha podido observar y deducir que los factores de Composición y de Comunicación son considerados por el estándar en todos los niveles de madurez.

Por el contrario, la Gestión del Equipo y la Coordinación, factores importantes para obtener el éxito en una iniciativa de mejora del proceso software, no son considerados por el estándar con el mismo nivel de detalle. En nuestra opinión se debería profundizar más en el análisis de estos dos factores en particular en toda iniciativa de mejora del proceso software.

Asimismo, aunque el factor de Motivación se ha considerado una característica esencial en un equipo de trabajo, el estándar ISO/IEC 15504 solamente considera este factor de manera explicita en las prácticas genéricas que se evalúan para medir el nivel de capacidad 5. Puesto que consideramos que este es un factor crucial, muchas veces descuidado, proponemos la consideración de este factor también en niveles inferiores de capacidad.

Con todo ello, para validar estos resultados preliminares es aún necesario demostrar que una mejora del trabajo en equipo puede dar como resultado una mejora del proceso software. Para

ello, el trabajo futuro deberá estar orientado a la aplicación real de un programa de mejora del proceso software en pequeñas y medianas empresas de desarrollo de software de acuerdo con el estándar internacional y centrado en los aspectos particulares de trabajo en equipo.

Agradecimientos : Las autoras de este trabajo quieren agradecer a la Comisión Interministerial de Ciencia y Tecnología (proyecto IN2GESOFT TIN2004-06689-C03-01) por respaldar esta línea de investigación.

Referencias

[1] Tarricone, P.; Luca, J.: "Employees, teamwork and social interdependence - a formula for successful business?". Team Performance Management: An International Journal, vol. 8, no. 3/4, (2002), pp. 54-59.

[2] Scarnati, J. T. "On becoming a team player". Team Performance Management: An International Journal, vol. 7, no. 1/2, (2001), pp. 5-10.

[3] Margerison, C. "Team Competencies". Team Performance Management: An International Journal, vol. 7, no. 7/8, (2001), pp. 177-122.

[4] Miller, E. "Teamwork on the Job - An Essential Ingredient to Success". IEEE-USA Today's Engineer Online", (2001). http://www.todaysengineer.org/Careerfocus/sept01te/sept01features/teamwork.html (disponible 07/12/2006)

[5] Gorla, N., Wah Lam, Y. "Who Should Work with Whom? Building Effective Software Project Teams". Communications of the ACM, vol. 47, no. 6, (2004), pp. 79-82.

[6] Mazhil, R. "Analysis of team effectiveness in software development teams working on hardware and software environment using Belbin Self-perception Inventory". Journal of Management Development, vol. 24, no. 8, (2005), pp. 738-753.

[7] Belbin, R. Management Teams: Why they succeed of fail. Elsevier Butterworth-Heinemann, Oxford (2004).

[8] Stevens, K., Henry, S. "Analysing Software Teams Using Belbin's Innovative Plant Role". http://www.radford.edu/~kstevens2/ISTall.pdf (disponible 18/12/2006)

[9] Evans, I. Achieving Software Quality through Teamwork. Artech House, Inc., Norwood (2004).

[10] Paulk, M., et al. The Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley (1995).

[11] CMU/SEI-2002-TR-008. Relating the Team Software Process℠ (TSP℠) to the Capability Maturity Model® for Software (SW-CMM®). Software Engineering Institute (2002).

[12] CMU/SEI-2003-TR-014. The Team Software ProcessSM (TSP℠) in Practice: A Summary of Recent Results. Software Engineering Intitute (2003).

[13] Noopur, D. "Using the TSP to Implement the CMM". CrossTalk. The Journal of Defense Software Engineering, (September 2002 Issue). http://www.stsc.hill.af.mil/crosstalk/2002/09/davis.html (disponible 10/07/2006)

[14] Hefley, B., Schwalb, J., Pracchia, L. "AV-8B's Experience Using the TSP to Accelerate SW-CMM Adoption". CrossTalk. The Journal of Defense Software Engineering, (September 2002 Issue). http://www.stsc.hill.af.mil/crosstalk/2002/09/hefley.html (disponible 27/12/2006).

[15] Pracchia, L. "The AV-8B Team Learns Synergy of EVM and TSP Accelerates Software Process Improvement". CrossTalk. The Journal of Defense Software Engineering, (January 2004 Issue). http://www.stsc.hill.af.mil/crosstalk/2004/01/0401pracchia.htm. (disponible 27/12/2006).

[16] Amengual, E.; Mas A. "A New Method of ISO/IEC TR 15504 and ISO 9001:2000 Simultaneous Application on Software SMEs". Proceedings of the Joint ESA - 3rd International SPICE Conference on

Process Assessment and Improvement, (March 2003), pp. 87-92.

[17] Mas, A.; Amengual, E. "A Method for the Implementation of a Quality Management System in Software SMEs". Software Quality Management XII. New Approaches to Software Quality, The British Computer Society, Great Britain (2004).

[18] Mas, A.; Amengual, E. "ISO/IEC 15504 Adaptation for Software Process Assessment in SMEs". Proceedings of the International Conference on Software Engineering Research and Practice, (June 2003), pp. 693-697.

[19] Blair, G.M. "Groups that Work". Engineering Management Journal, vol. 1, no. 5, (1991), pp. 219-223.

[20] MSF Team Model v. 3.1. Microsoft Corporation (2002).

[21] Blair, G.M. "The Human Factor". Engineering Management Journal, vol. 2, no. 5, (1992), pp. 219-223.

[22] ISO/IEC 15504-5:2006. Information technology -- Process Assessment -- Part 5: An exemplar Process Assessment Model. International Organization for Standardization (2006).

[23] ISO/IEC 15504-2:2003. Information technology -- Process assessment -- Part 2: Performing an assessment. International Organization for Standardization (2003).

PMO: Un sistema de gestión del conocimiento en la gestión de proyectos software

Fran J. Ruíz-Bertol
Dpto. Informática e Ingeniería de Sistemas
Universidad de Zaragoza
Centro Politécnico Superior
50018 Zaragoza
franjr@unizar.es

Javier Dolado
Dpto. Lenguajes y Sistemas Informáticos
Universidad del País Vasco
Facultad de Informática
20018 San Sebastián
javier.dolado@ehu.es

Resumen

La gestión de proyectos ha sido durante muchos años un área de conocimiento reservada a expertos y profesionales. Sin embargo, dicha experiencia está siendo recopilada en multitud de artículos y libros, e implementada en varios sistemas software a la que cualquier puede acceder. Debido al creciente interés en las tecnologías informáticas para la gestión del conocimiento, actualmente disponemos de suficientes técnicas, herramientas, y habilidades para poder desarrollar una representación del conocimiento de la gestión de proyectos, añadiendo todas las características inherentes a los sistemas basados en el conocimiento. En este artículo, se presenta *Project Management Ontology* (PMO), una ontología de dominio, que recoge tanto la estructura común para la gestión de proyectos, como la información asociada para poblar la ontología. PMO se ha creado de forma modular, permitiendo aplicar la gestión de proyectos a distintas areas de conocimiento, a través de la unión de ontologías.

1. Introducción

En la actualidad existe un creciente interés sobre la adecuada gestión del conocimiento en las distintas áreas de conocimiento. De hecho, en el mundo del desarrollo software, es muy importante gestionar adecuadamente dicho conocimiento, tanto el que se da de manera explícita como el tácito o implícito. El primero se define como "*una herramienta de gestión para aprovechar la manipulación del conocimiento de la organización. groupwares, intranets, servidores de listas, repositorios de conocimiento, gestión de bases de datos y redes de acción del conocimiento permiten compartir la dicha gestión del conocimiento*"[12]. El segundo, más difícil de capturar, se basa más bien en la experiencia, guiado por el contexto, y por lo general, reside en los individuos.

En los proyectos software, dicho conocimiento es fundamental por muchas razones: registro histórico, lecciones aprendidas, explotación de datos, toma de decisiones, seguimiento del proyecto, metodologías utilizadas, estimación y planificación, asignación de recursos, etc. Por ello, es necesario capturar y gestionar el conocimiento disponible en un formato y representación adecuados. Esto puede realizarse para la mayoría de áreas de conocimiento, donde pueden definirse por experto un conjunto de conceptos, aserciones, reglas e inferencias sobre dicha información. Ésta información puede guardarse utilizando alguna de las representaciones de conocimiento existentes.

La representación del conocimiento tiene varias funciones: sirve como modelo de la realidad, establece hechos sobre el mundo real, determina un conjunto de aserciones para inferir y razonar sobre dicho modelo, proporciona una

organización adecuada del conocimiento, y facilita un lenguaje que expresar el conocimiento humano [3].

Una manera razonable de determinar dicho conocimiento es la utilización de ontologías, que proporcionan "*una especificación formal y explícita de una conceptualización compartida*"[13], esto es, una representación declarativa de conceptos, estructuras de datos, relaciones, aserciones, reglas y restricciones que representan un modelo abstracto y simplificado de la realidad. Las ontologías se expresan comúnmente utilizando ontologías específicas de dominios. Estas ontologías tienen dos características: proporcionan una representación explícita de un modelo conceptual sobre un dominio determinado; y dicho modelo establece una representación compartida y consensuada sobre el conocimiento para dicho dominio.

En la actualidad, existen muchas ontologías de dominio y sistemas basados en el conocimiento que abarcan un amplio conjunto de áreas de conocimiento en diversos dominios. Se pueden encontrar información sobre dichas ontologías en varios repositorios de ontologías [1][4][10]. Sin embargo, aún no existe una representación formal y explícita para la gestión de proyectos.

En este artículo se presenta *Project Management Ontology* (PMO), una ontología de dominio que representa un modelo formal de los procesos, actividades, herramientas y técnicas específicas de la gestión de proyectos. PMO proporciona una descripción completa de los términos fundamentales y características inherentes al manejo de la información asociada a la gestión, seguimiento, control y dirección de los proyectos, así como de los procesos, relaciones, restricciones y aserciones sobre los datos de proyectos.

El artículo está dividido de la siguiente manera: la sección 2 describe *Project Management Ontology* (PMO), detallando las principales características de esta ontología, el proceso de desarrollo y sus componentes. En la sección 3 se explica cómo PMO puede ser integrada con otras áreas de conocimiento utilizando técnicas de mapeado y fusión. En la sección 4 se exponen las conclusiones y el trabajo futuro.

2. *Project Management Ontology* (PMO)

La gestión de proyectos se define como el conjunto de herramientas, técnicas, conocimiento y habilidades aplicadas a un proyecto para cumplir un conjunto de requisitos, estándares, especificaciones y objetivos que llevan a completar dicho proyecto. La gestión por proyectos se llevan a cabo en varios ámbitos, incluyendo entre ellos a su aplicación en arquitectura e ingeniería, industria química, desarrollo software o en el ámbito de la investigación. En todos estos ámbitos, es de vital importancia la gestión de proyectos, ya que proporciona un conjunto guiado de procesos que permiten realizar un control y evaluación de las tareas a desarrollar para conseguir el producto o servicio final.

Para gestionar los proyectos, se dispone de varias herramientas centradas en capturar una parte de la información del proyecto. Sin embargo, estas herramientas se utilizan de una manera aislada -válida únicamente para las personas que utilizan dichas herramientas-, o como un conjunto integrado de aplicaciones. Por lo tanto, es complicado compartir este conocimiento con otras organizaciones o incluso, dentro de una misma organización que utilice distintas aplicaciones para la gestión de proyectos, incluso si existe la posibilidad de importar/exportar la información. Por ello, es necesario definir una representación del proyecto que pueda ser utilizada por dichas aplicaciones, pero que también permita la interoperabilidad entre éstas. En este sentido, se considera necesario establecer una representación del conocimiento que permita capturar y gestionar adecuadamente la información del proyecto. Para éste propósito, lo más adecuado es la utilización de ontologías de dominio.

Para desarrollar esta ontología, ha sido necesario plantearse las distintas opciones para capturar dicho conocimiento: (i) hablar con expertos en la gestión de proyectos, (ii) capturar el conocimiento directamente de las aplicaciones de gestión de proyectos, o bien, (iii) obtener de alguna fuente de información un modelo descriptivo y consensuado del

conocimiento. La primera opción es difícil de lograr, ya que parte de dicho conocimiento es tácito y por lo tanto, no hay forma de modelarlo adecuadamente. En la segunda opción, el modelo de desarrollo seguido por los desarrolladores está centrado en la aplicación, por lo que el modelo de datos únicamente tiene los parámetros suficientes para que la aplicación funcione, sin que esté definido el conocimiento para la gestión de proyectos de una manera completa. Finalmente, para la tercera opción, existe una gran cantidad de literatura, tanto en forma de libros como en forma de artículos de investigación. Entre todos ellos, se ha seleccionado *Project Management Body of Knowledge*® (PMBOK) [8], ya que proporciona una documentación exhaustiva en el área de la gestión de proyectos que ha sido desarrollado por expertos en el área y en áreas adyacentes.

Para capturar y gestionar dicho conocimiento, se ha desarrollado *Project Management Ontology* (PMO), un conjunto de ontologías que abarcan los principales procesos, conceptos y relaciones de la gestión de proyectos. Este sistema basado en el conocimiento proporciona la primera representación formal de conocimiento en el dominio de la gestión de proyectos. Para su desarrollo, ha sido fundamental desarrollarlo de una manera modular y estructurada, de forma que otras ontologías o sistemas basados en el conocimiento puedan unirse a PMO para crear una ontología específica de la gestión de proyectos en el dominio aplicado. PMO está compuesta de:

- Una taxonomía que define la estructura de un proyecto. Esta taxonomía proporciona una jerarquía basándose en la estructuración de los proyectos, así como en la definición de los principales términos. Esta taxonomía está formada principalmente por relaciones del tipo *isA* (subclases) o *has* (composición), que define esencialmente las partes en que se puede dividir y estructurar un proyecto y sus componentes.
- Un completo vocabulario que define conceptos específicos de la gestión de proyectos, y que pueden ser aplicables a la mayoría de áreas de conocimiento. Este vocabulario ha sido obtenido del glosario de términos del PMBOK [8].
- Un sistema basado en el conocimiento (KBS) que contiene el conocimiento experto de cómo gestionar un proyecto de manera adecuada. Esto incluye la estructura, el contenido semántico, y los procesos e instancias necesarias que proporcionan una guía para la dirección.
- Un conjunto de *slots* o propiedades asociadas a uno o varios de los conceptos presentes en la taxonomía o el vocabulario.
- El conjunto de relaciones entre conceptos definidos tanto a nivel de taxonomía, como entre los distintos componentes de PMO (utilizando las propiedades *owl:equivalentClass* y *owl:sameAs*). Estas relaciones están basadas en la propia definición de los concpetos.

Se puede observar la estructura básica de PMO en la Figura 1. Cada uno de los componentes mostrados en esta figura representa una ontología. En PMO actualmente se han definido cinco ontologías: *PM-Cost*, *PM-Process*, *PM-Planning*, *PM-Organization*, y la ontología núcleo *PM-Core*.

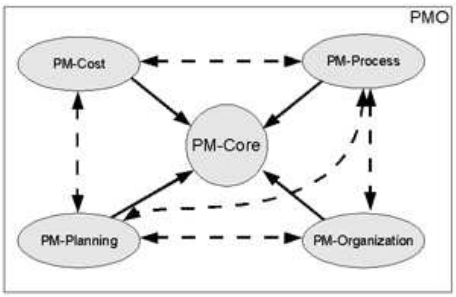

Figura 1: Componentes de *Project Management Ontology*

En la Figura 1, se pueden diferenciar dos tipos de relaciones: las indicadas con línea continua, y las indicadas con punteado. Las flechas continuas indican una correspondencia

directa entre clases que representan un mismo concepto utilizando la propiedad de OWL *owl:sameAs*, como por ejemplo los conceptos que representan *Actividad* y *Paquete de Trabajo*. Las flechas punteadas representan conceptos relacionados entre sí a través de los conceptos definidos en *PM-Core*.

Para el proceso de desarrollo se ha seguido la metodología recomendada por Noy y McGuiness [6]. Este proceso de desarrollo consiste en un conjunto de pasos propuestos por los autores para la creación de ontologías de dominio: (i) determinación del dominio y ámbito de la ontología; (ii) consideración de reutilización de ontologías ya existentes; (iii) enumeración de los conceptos y términos clave; (iv) definición de la estructura de clases y atributos de cada una de las clases; (v) definición de las restricciones sobre los atributos; y (vi) completar la ontología, poblándola con instancias o individuos.

Para crear una vista gestionable del dominio de gestión de proyecto, se ha dividido PMO en varios componentes u ontologías, cada uno de ellos proporcionando una visión parcial de una parte del conocimiento en la gestión de proyectos. Esta estructura se ha definido en base a las distintas partes diferenciadas en las que se compone el área de conocimiento (coste, planificación, riesgos, requisitos, aseguración de la calidad, etc.). PMO tiene los siguientes componentes:

- *PM-Core*. El conjunto de conceptos, relaciones, axiomas y atributos que forman la base para la gestión de proyectos. En esta ontología se incluyen conceptos como proyecto, fase, entregables, productos, o actividades.

- *PM-Process*. Esta ontología representa el conjunto de procesos y grupos de procesos de recomendada aplicación para guiar un proyecto. Esta ontología proporciona principalmente el conocimiento recogido en el PMBOK [8].

- *PM-Organization*. Esta ontología proporciona la estrucutra organizativa del proyecto, definiendo los conceptos de equipo, persona, atribuciones, habilidades, asignaciones, etc. El enfoque tomado para el desarrollo de esta ontología ha sido la división desde el punto de vista de la gestión para la organización que desarrolla el proyecto, el desarrollo de los equipos, y los actores que forman parte de dicho proyecto.

- *PM-Cost*. Esta ontología incluye todos aquellos conceptos, atributos y relaciones asociados a la gestión de costes, tanto monetarios como de esfuerzo. También incluye conceptos relacionados con las estimaciones y presupuestos.

- *PM-Planning*. Finalmente, esta ontología desarrolla toda la parte de gestión de la planificación, calendario y seguimiento de un proyecto.

Aún están en proceso de desarrollo ontologías adicionales que completen las partes de conocimiento no incluidas en este primer desarrollo de PMO, ya que se consideraron en un primer momento como secundarias en la definición del domino. Entre estas partes del conocimiento no desarrolladas están los riesgos, los requisitos, la aseguración de la calidad y la gestión de las comunicaciones. Una vez desarrolladas este conjunto secundario de ontologías pasarán a formar parte de PMO, utilizando la unión de ontologías, con el objetivo de completar el conocimiento del dominio.

2.1. *PM-Core*

PM-Core es la principal ontología de PMO. Todas las demás ontologías en PMO están directamente relacionadas con *PM-Core*, como se puede observar en la Figura 1. *PM-Core* contiene el vocabulario y la estructura básica para la definición de un proyecto genérico, en cualquier ámbito de aplicación. El principal concepto definido en la ontología es el *Proyecto*, que tiene asociado una serie de atributos inherentes, como su nombre, descripción, procesos a utilizar, criterios de calidad o hitos. Así mismo, un proyecto puede ser parte de un *Programa* o un *Portafolio*. En el proceso de división hacia artefactos más manejables, se

pueden definir distintos tipos de *Componentes de proyecto* (clase abstracta): *Fases* (o subfases), *Entregables*, o *Paquetes de trabajo*, siguiendo las recomendaciones de estructuración del trabajo del proyecto [9]. Cada uno de estos componenetes pueden combinarse entre sí mediante las relaciones definidas en la propia ontología, para formar la *Estructura de Descomposición del Trabajo* (WBS).

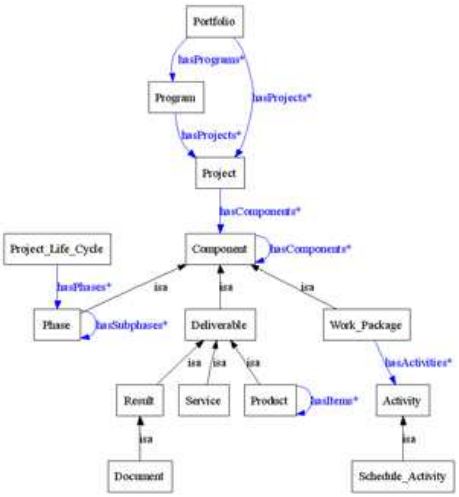

Figura 2: Vista simplificada de la jerarquía de clases de *PM-Core*

En la Figura 2, se puede observar la jerarquía de clases definida en *PM-Core*. Tal y como se define en [9], "*una Estructura de Descomposición del Trabajo es una descomposición jerárquica orientada a ser entregada del trabajo a ser ejecutado por el equipo del proyecto para cumplir los objetivos del proyecto y crear los entregables solicitados*". Los elementos terminales son principalmente entregables o paquetes de trabajo, que se dividen a su vez en actividades. Las actividades representan los elementos gestionables más pequeños que pueden asignarse directamente a un equipo del proyecto o una persona.

2.2. *PM-Process*

PM-Process es la ontología que contiene el modelo de procesos de gestión del PMBOK [8]. En esta ontología se incluyen aquellos grupos de procesos en los que está dividido el cuerpo del conocimiento en la gestión del proyecto: procesos de iniciación, planificación, ejecución, control y monitorización, y finalización.

Estos grupos de procesos son necesarios para cualquier proyecto, ya que proporciona un proceso guiado, siendo necesario que se ejecuten siguiendo la misma secuencia para todos los proyectos, independientemente del ciclo de vida utilizado o el proceso de desarrollo software seleccionado. Cada uno de estos grupos de procesos contiene una serie de procesos de gestión interrelacionados entre sí, tal y como se define en el PMBOK. Esto incluye una descripción completa del proceso de gestión, así como las entradas, salidas, herramientas y técnicas necesarias para llevar a cabo dicho proceso. El equipo de gestión del proyecto será el encargado de seleccionar el subconjunto adecuado de estos procesos para llevar a cabo el conjunto de actividades del proyecto.

Sin embargo, *PM-Process* no está restringido a la definición de una serie de clases, relaciones y propiedades para almacenar el conocimiento sobre los procesos de gestión, sino que también proporciona un sistema basado en el conocimiento, ya que la ontología se ha completado con un conjunto de instancias que definen los procesos con la información obtenida del PMBOK [8].

Este conocimiento tiene el objetivo de proporcionar una guía a la hora de definir el conjunto de actividades presentes en *PM-Core*, pero también de adecuar la gestión de estas actividades al proyecto específico. Esta ontología representa únicamente los procesos de gestión, por lo que no incluye otros procesos que no son específicamente procesos de gestión, por ejemplo, procesos software, que podrían estar definidos en otra ontología.

2.3. *PM-Organization*

En un sentido estricto, la organización y los recursos humanos no es una parte constituyente de la gestión de proyectos, aunque sí que es una pieza fundamental para el éxito del proyecto. De hecho, algunos procesos de

gestión están directa o indirectamente relacionados a la organización (por ejemplo, todos aquellos procesos de la Gestión de Recursos Humanos del Proyecto presentes en el PMBOK).

Durante el desarrollo de esta ontología se ha utilizado parte el trabajo ya desarrollado en las ontologías de organización disponibles en [7], que describen las principales características de una organización, incluyendo metas, jerarquía, roles, puestos desempeñados, personas, equipos, etc. La utilización del conocimiento disponible en estas ontologías ha sido de gran utilidad para crear y adaptarlas en *PM-Organization* a un enfoque de gestión de proyectos.

En *PM-Organization* se han definido los conceptos básicos de una organización, como organización, persona, empleado, puesto o habilidades, y se ha extendido añadiendo conceptos más relacionados a la gestión del proyectos, como equipo de proyecto, miembro del equipo, gestor/director o equipo virtual. Finalmente se han redefinido o agregado nuevos términos conducentes a su adaptación a las actividades de gestión, como por ejemplo, habilidades de gestión, responsabilidades de gestión, competencias o comunicaciones.

Esta ontología es una base del conocimiento que puede unirse a *PM-Core* para proporcionar una definición completa del proyecto, tanto desde el punto de vista structural como desde el organizativo. La estructura de la taxonomía asociada a *PM-Organization* puede observarse en la figura 3.

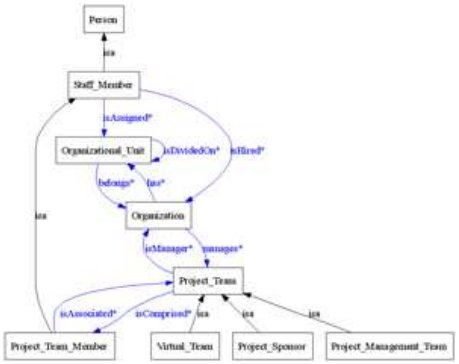

Figura 3: Estructura de la jerarquía de clases de *PM-Organization*

2.4. *PM-Planning*

La ontología *PM-Planning* contiene el conocimiento necesario para la representación de la planificación, calendario, estimaciones, asignación, seguimiento, control, gestión de cambios y finalización de las actividades del proyecto.

La unidad utilizada en esta ontología es principalmente el tiempo, aunque también están consideradas otras unidades de medida, como los recursos y el esfuerzo. Debido a que ya existen representaciones del tiempo en forma de ontologías, se ha considerado conveniente la utilización de *OWL Time Ontology* [14] para ayudar al desarrollo de la ontología *PM-Planning*. De hecho, todas las medidas de duración e intervalos de tiempo han sido definidas en función de esta ontología. En *PM-Planning* se pueden diferenciar dos tipos de medidas:

- Las propiedades que están directamente relacionadas con la planificación. Entre estas propiedades se incluyen aquellos valores necesarios para las estimaciones y seguimiento del proyecto, como por ejemplo, *Fecha de Inicio Temprana*, *Fecha de Inicio Tardía*, *Fecha de Finalización Temprana* y *Fecha de Finalización Tardía*, utilizados en el Método del Camino Crítico [5], o simplemente otras propiedades directamente asociadas al cumplimiento de plazos de las actividades (por ejemplo, *Fecha Actual*, *Fecha de Finalización Actual*, *Fecha de Finalización Estimada*, etc.).

- Otros conceptos que definen estructuras de planificación y estimación más complejas como las Líneas de Base.

En *PM-Planning* se ha decidido incluir la definición temporal para distintos husos horarios y calendarios, ya que los proyectos, especialmente aquellos que trabajan con equipos virtuales, son desarrollados generalmente en distintas localizaciones. *PM-Planning* está

muy relacionada con *PM-Cost* a través de *PM-Core*, ya que ambas comparten varios de los términos de alto nivel, como por ejemplo, la *Línea de Base* o la *Secuencia del Camino Crítico*, entre otros.

2.5. *PM-Cost*

El componente *PM-Cost* representa el conjunto de conceptos relativos al presupuesto, planificación, estimación y control de los costes. Se ha diferenciado *PM-Cost* de *PM-Planning* ya que ambos componentes representan diferentes términos: el coste del proyecto determina generalmente un valor monetario del proyecto, mientras que el calendario engloba los aspectos temporales del proyecto, comúnmente más utilizados para medir la progresión del proyecto. De hecho, las organizaciones generalmente tienen dos aplicaciones separadas para las estimaciones de coste/presupuesto y el calendario. Sin embargo, ambas ontologías determinan una información fundamental en la gestión de proyectos. De hecho, determinar buenas estimaciones del presupuesto y del calendario son factores fundamentales para el éxito del proyecto [15].

La ontología *PM-Cost* está principalmente expresada en términos de coste monetario dada una cierta moneda (por ejemplo, si el proyecto se está realizando conjuntamente entre Europa y EE.UU., los equipos del proyecto trabajarán tanto en Euros como en Dólares). Debido a que el proyecto generalmente está supeditado a un presupuesto definido, las unidades a utilizar para expresar el coste no deberían cambiar durante dicho proyecto, o al menos no deberían estar sujetas a los cambios monetarios que se produzcan. Este hecho está capturado en la ontología, donde se establecen distintas unidades monetarias, y un tipo de datos con funciones embebidas para hacer transparente el coste del proyecto.

3. Integración de PMO

En la actualidad, una de las principales características de las ontologías es facilitar la interoperabilidad con otra información, ontologías o aplicaciones. Durante el proceso de desarrollo de PMO, se ha decidido desarrollar una definición genérica del área de conocimiento de la gestión de proyectos, de tal manera que pueda hacerse extensible a otras áreas de conocimiento diferentes. De hecho, se puede afirmar que esta característica es fundamental para la integración del conocimiento de la gestión de proyectos con otras áreas de conocimiento, como la ingeniería del software o la arquitectura. Así, PMO ha sido desarrollada utilizando un enfoque modular, para poder facilitar la integración con otras formas de conocimiento (ontologías, bases de datos, sistemas de gestión del conocimiento, aplicaciones, etc.).

Por una parte, todos los componentes de PMO están altamente integrados. Este hecho puede observarse en la Figura 1, donde las dependencias entre todos los componentes de PMO están relacionados de una u otra manera. Por ejemplo, la ontología *PM-Process* puede ser fácilmente definida independientemente de las demás ontologías, pero los conceptos y propiedades presentes en ella están fuertemente relacionados en la manera de distribuir el trabajo al equipo del proyecto (presente en *PM-Organization*). De la misma manera, se pueden encontrar relaciones similares en las demás ontologías de PMO. Actualmente, se está desarrollando nuevos componentes para capturar mayor información en áreas asociadas a la gestión de proyectos, como los riesgos, las comunicaciones o la calidad del proyecto.

Por otra parte, es recomendable integrar este sistema basado en el conocimiento con otras ontologías existentes, para extender y enriquecer el ámbito de actuación de PMO. Esto se puede realizar mediante procesos de unión y mapeado de ontologías [2]. Supongamos que se desea unir PMO con una ontología que modela el conocimiento en Ingeniería del Software. Se puede realizar un mapeado de ambas ontologías siempre que los conceptos estén descritos de similar manera o se puedan encontrar conceptos equivalentes en ambas ontologías. Por ejemplo, en *PM-Process* se ha definido el concepto de *Proceso*, que tiene

una subclase denominada *Procesos de Gestión*, que especifica los procesos que se recomiendan para una gestión eficiente de los proyectos. En otra ontología, asociada al conocimiento en la Ingeniería del Software, es altamente probable que nos encontremos con una serie de conceptos que definan los procesos software. Éstos procesos software dependerán del ciclo de vida seleccionado, pero también de las políticas aplicadas en la organización, la experiencia del gestor del proyecto, etc. Se puede afirmar que si dicha ontología o representación de conocimiento existe, podría definirse una mapeado entre ambas ontologías para la unión del conocimiento en la gestión de proyectos al desarrollo del proyecto software.

En la Figura 4, dadas dos ontologías, PMO y otra que represente el conocimiento sobre los procesos software, ambas pueden ser unidas utilizando un mapeado del conocimiento. De hecho, en la actualidad existen varios trabajos abiertos sobre el mapeado de ontologías [2], que incluyen entornos de trabajo para el mapeado, métodos, herramientas, traductores, mediadores y técnicas.

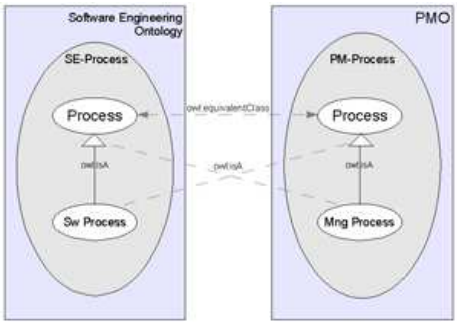

Figura 4: Un ejemplo de unión de dos ontologías de áreas de conocimiento diferentes

4. Conclusiones

En este artículo se ha presentado *Project Management Ontology*, un sistema basado en el conocimiento para la captura y gestionar el conocimiento sobre la gestión de proyectos. Esta representación del conocimiento proporciona una visión semántica de la información del proyecto utilizando el enfoque de gestión.

Este trabajo propone un conjunto de ontologías básicas que son necesarias para comenzar a capturar y salvar la información de gestión de los proyectos en un formato independiente de las aplicaciones utilizadas, facilitando así mismo el proceso de interoperabilidad e intercambio de información. Aunque el proceso de desarrollo de ontologías es duro y bastante largo, la necesidad de proporcionar y obtener el conocimiento disponible sobre la gestión de proyectos es fundamental para gestionar de una manera cada vez más eficiente. De hecho, en la actualidad, existe una gran cantidad de grupos de investigación dedicados a la gestión del conocimiento.

Así mismo, en este artículo se han mostrado las principales características de *Project Management Ontology* (PMO). PMO está compuesta de varias ontologías, donde cada una desarrolla una parte del conocimiento de la gestión de proyectos. Esta estructuración por partes de conocimiento, ha permitido trabajar individualmente con cada una de ellas (dividiendo el problema de la representación del conocimiento en partes más pequeñas), lo que ha facilitado su posterior proceso de integración. Así mismo, en la actualidad se está tratando de unir esta ontología con ontologías de otras áreas de conocimiento.

5. Agradecimientos

El desarrollo de la investigación asociada a este artículo ha sido posible gracias a la financiación del proyecto de investigación TIN2004-06689-C03-01 por el Ministerio de Educación y Ciencia.

Referencias

[1] DAML Ontology Library [online], *http://www.daml.org/ontologies/*, Accedido: 16/04/2007.

[2] Ehrig M., Sure Y., *Ontology Mapping - An Integrated Approach*, The Semantic Web:

Research and Applications, First European Semantic Web Symposium, (ESWS 2004), 2004.

[3] Gasevic D., Djuric D., Devedzic V., *Model Driven Architecture and Ontology Development*, Springer-Verlag, 2006.

[4] KBS/Ontology Projects Worldwide [online], *http://www.cs.utexas.edu/users/mfkb/related.html*, Accedido: 16/04/2007.

[5] Kerzner H., *Project Management: A Systems Approach to Planning, Scheduling, and Controlling. 8th ed.*, Willey, 2004.

[6] Noy N.F., McGuiness D.L., *Ontology Development 101: A Guide to Creating your First Ontology*, 2001.

[7] Organization Ontology [online], *http://www.cs.umd.edu/projects/plus/SHOE/onts/org1.0.html*, Accedido: 16/04/2007.

[8] Project Management Institute, *A guide to project management body of knowledge: PMBOK guide. 3rd ed.*, Project Management Institute, 2004.

[9] Project Management Institute, *Practice Standard for Work Breakdown Structures. 2nd ed.*, Project Management Institute, 2006.

[10] Protégé Wiki: Protégé Ontology Library [online], *http://protege.cim3.net/cgi-bin/wiki.pl?ProtegeOntologiesLibrary*, Accedido: 16/04/2007.

[11] Quantitative Software Management's SLIM® (Software LIfecycle Management) [online], *http://www.qsm.com/products.html*, Accedido: 16/04/2007.

[12] Scarborough H., Swan J., Preston J., *Knowledge Management: a Literature Review: Issues in People Management*, Institute of Personnel and Development (Londres), 1999.

[13] Studer R., Benjamins V.R., Fensel D., *Knowledge Engineering. Principles and Methods*, Data and Knowledge Engineering 25 (1-2) 1998.

[14] Time Ontology in OWL [online]. W3C Working Draft 27 September 2006, *http://www.w3.org/TR/owl-time/*, Accedido: 16/04/2007.

[15] White D., Fortune J., *Current practice in project management - an empirical study*, International Journal of Project Management 20, 1-11, 2001.