Revista Científica Mundo de la Investigación y el Conocimiento

Miryan Dorila Iza Carate [a]

Gestión del conocimiento en ingeniería de software

Knowledge Management in Software Engineering

Revista Científica Mundo de la Investigación y el Conocimiento. Vol. 2 núm.4, Octubre, ISSN: 2588-073X, 2018, pp. 32-47

DOI: 10.26820/recimundo/2.(4).octubre.2018.32-47

URL: http://www.recimundo.com/index.php/es/article/view/338

Editorial Saberes del Conocimiento

Recibido: 15/07/2018 Aceptado: 05/08/2018 Publicado: 30/10/2018

Correspondencia: miry_doris@yahoo.es

a. Magister en Ingeniería de Software; Diploma Superior en Gestión para el Aprendizaje Universitario; Ingeniera en Sistemas e Informática; Docente de la Universidad Técnica de Cotopaxi; miry_doris@yahoo.es

RESUMEN

La Gestión del Conocimiento es una disciplina emergente que promete capitalizar capital intelectual de las organizaciones. El concepto de conocimiento está lejos de ser nuevo y frases que contienen la palabra conocimiento como “bases de conocimiento” e “ingeniería del conocimiento” han existido por un tiempo. La comunidad de inteligencia artificial (IA). Por ejemplo, ha tratado durante mucho tiempo la representación, el almacenamiento y la aplicación del conocimiento.

Entonces, la gestión del conocimiento es uno de esos conceptos promocionados que se elevan de manera rápida y ambiciosa. Afirma curar los dolores de cabeza organizacionales y luego falla y cae silenciosamente? O es un instrumento que realmente ayudará a las organizaciones a abordar algunos de los problemas que enfrentan mientras intentas alcanzar sus objetivos de negocio? En particular, es la gestión del conocimiento valiosa para las organizaciones de desarrollo de software? Qué tipo de problemas puede ayudar a resolver? Cómo se puede implementar un sistema de gestión de conocimiento para una organización de software? Cuáles son los factores de éxito? Este trabajo aborda las preguntas anteriores e intenta proporcionar respuestas que resultaron de una extensa investigación sobre el estado del arte y el estado de la práctica de este tema.

Palabras claves: Ingeniería de software, Gestión del conocimiento, Organizaciones.

ABSTRACT

Knowledge Management is an emerging discipline that promises to capitalize on the intellectual capital of organizations. The concept of knowledge is far from new and phrases that contain the word knowledge as "knowledge bases" and "knowledge engineering" have existed for a while. The artificial intelligence community (AI). For example, he has long dealt with the representation, storage and application of knowledge.

Then, knowledge management is one of those promoted concepts that rise quickly and ambitiously. Claims to cure organizational headaches and then fails and falls silently? Or is it an instrument that will really help organizations address some of the problems they face while trying to achieve their business objectives? In particular, is the management of valuable knowledge for software development organizations? What kind of problems can you help solve? How can you implement a knowledge management system for a software organization? What are the success factors? This paper addresses the above questions and attempts to provide answers that resulted from extensive research on the state of the art and the state of practice of this topic.

Keywords: Software engineering, Knowledge management, Organizations.

Introducción.

El primer argumento a favor de la gestión del conocimiento en ingeniería de software es que es un actividad intensiva en humanos y conocimiento (Birk, Surmann, & Althoff, 1999). Similar a otros sectores, consultoría, derecho, banca de inversión y publicidad, el principal activo de una organización consiste en su capital intelectual. El desarrollo de software es un "tipo de proceso" donde cada persona involucrada tiene que tomar una gran cantidad de decisiones, cada una de ellas con varias opciones posibles, a diferencia de un proceso de "producción" o "fabricación" donde, una vez que se toma una decisión, muchos trabajadores pueden realizar tareas sin tener que tomar decisiones posteriores. Por ejemplo, una empresa debe seleccionar qué productos desarrollar; un gerente de proyecto debe seleccionar el personal y debe planificar un proyecto, lo que implica seleccionar un proceso y un conjunto de métodos y técnicas a utilizar; un diseñador debe seleccionar un algoritmo eficiente; un programador tiene que decidir sobre una función, o variables a usar; y un probador debe seleccionar un conjunto de casos de prueba. ¿Cómo todas estas personas toman sus decisiones? ¿En qué se basan? La mayoría de las veces, los tomadores de decisiones confían en el conocimiento personal y experiencia, en su "instinto". Pero a medida que los proyectos de desarrollo de software crecen la disciplina pasa de la artesanía a la ingeniería, se convierte en una actividad grupal donde los individuos necesitan comunicarse y coordinarse. El conocimiento individual tiene que ser compartido y aprovechado a nivel de proyecto y organización, y esto es exactamente lo que la gestión del conocimiento propone, desmitifica al héroe individual y cambia el enfoque a creatividad colectiva, explotando la idea de comportamiento emergente “ninguno de nosotros es tan inteligente como todos nosotros” (Bennis & Biederman, 1998).

En el desarrollo de software se pueden identificar dos tipos de conocimiento:

1. El conocimiento incorporado en los productos (artefactos), ya que son el resultado de actividades intelectuales altamente creativas.
2. meta-conocimiento, es decir, conocimiento sobre los productos y procesos.

Algunas de las fuentes de conocimiento (por ejemplo, los artefactos) se almacenan por defecto en medios electrónicos por la naturaleza misma del desarrollo de software, por lo que el uso de la gestión del conocimiento se facilita en esta industria. Sin embargo, esta fuente se puede ver como datos e información en bruto y busca convertir datos en información e información en el conocimiento. El problema más importante es, sin embargo, que sólo una fracción de todo el conocimiento relacionado con el software es capturada y hecho explícito. La mayoría del conocimiento es tácito, reside en el cerebro de los empleados, este hecho hace que el intercambio de conocimientos y retener el conocimiento sea un desafío.

El propósito de este informe es describir el estado del arte de la Gestión del Conocimiento en Ingeniería de Software. Se enfrenta al desarrollo de software y muestra cómo la gestión del conocimiento puede ayudar a resolverlos.

Metodología.

Esta investigación está enfocada en el estudio y aplicación de la gestión del conocimiento en el sector de la ingeniería de software como una manera de promover una adecuada toma de decisiones incluyendo los distintos factores que esto conlleva.

La revisión se ha centrado en textos, documentos y artículos científicos publicados disponibles en la web, considerando que aquella herencia de la globalización nos permite acceder a mayor y mejor información a través de las herramientas tecnológicas. El motor de búsqueda ha sido herramientas académicas de la web que direccionan específicamente a archivos con validez y reconocimiento científico, descartando toda información no confirmada o sin las respectivas referencias bibliográficas.

Resultados.

La ingeniería de software es un negocio complejo que involucra a muchas personas que trabajan en diferentes fases y actividades. Los constantes cambios tecnológicos hacen que el trabajo sea dinámico: Se resuelven nuevos problemas y se crean nuevos conocimientos cada día. El conocimiento en la ingeniería de software es diversa y sus proporciones son inmensas y en crecimiento. Las organizaciones tienen problemas para hacer un seguimiento de lo que es este conocimiento, dónde está y quién lo tiene. Una forma estructurada de gestionar el conocimiento y tratar el conocimiento y sus propietarios como activos valiosos podrían ayudar a las organizaciones a aprovechar el conocimiento que poseen.

La ingeniería de software es un negocio intensivo en conocimiento y como tal, podría beneficiarse de las ideas de gestión del conocimiento. Lo importante La pregunta es, sin embargo, ¿dónde reside el conocimiento en la ingeniería de software? Está claro que la ingeniería de software implica una multitud de tareas que requieren un uso intensivo del conocimiento: Analizar los requisitos de los usuarios para los nuevos sistemas de software, identificar y aplicar mejor Prácticas de desarrollo de software, recogiendo experiencia sobre planificación de proyectos y gestión de riesgo entre otros (Birk, Surmann, & Althoff, 1999)

Identificamos tres categorías principales para tareas de ingeniería de software:

1. Tareas realizadas por un equipo que se enfoca en desarrollar un producto de software basado en requerimientos del cliente. Esto representa la tarea central de cualquier organización de desarrollo de software. El líder del equipo (jefe de proyecto) es responsable de garantizar que el trabajo se complete a tiempo y dentro del presupuesto y posea la intención funcionalidad y calidad. La ingeniería de software está orientada a documentos y lo que es producido durante el proyecto es un conjunto de documentos tales como contratos, planes de proyecto, requisitos y especificaciones de diseño, código fuente, planes de prueba y documentos relacionados. Estos documentos no son solo productos de trabajo. También hay información adicional incrustada dentro de ellos:

- Durante el proyecto se documentan las decisiones.
- Después de la finalización del proyecto, contienen la historia del mismo. Los documentos pueden ser reutilizados de diferentes maneras para el próximo proyecto para que las personas pueden aprender de ellos, analizando las soluciones a diferentes problemas que estos documentos reflejan.

2. Tareas que se centran en mejorar la capacidad de un equipo para desarrollar un producto de software (Eso es mejorar tareas en la primera categoría). Aquí podemos incluir tareas que podrían llevarse a cabo durante y poco después del proyecto. La razón de realizar estas tareas es garantizar que el conocimiento potencial adquirido en el proyecto no está perdido. Aquí se incluyen todas las formas de lecciones aprendidas y análisis post-mortem que identifican lo que salió bien o mal en el proyecto. También se incluyen análisis de datos del proyecto, por

ejemplo, comparaciones de presupuestos y costos reales, esfuerzo estimado y real, tiempo del calendario planificado y real. Las tareas en esta categoría intentan recopilar y crear conocimiento sobre un proyecto en particular. Los resultados de esta actividad son útiles por sí mismos, pero también pueden ser la base para un mayor aprendizaje. Se pueden almacenar en repositorios y base de datos de experiencias.

3. **Tareas que se centran en mejorar la capacidad de una organización o industria para desarrollar software.** Esta categoría representa actividades que analizan resultados de varios proyectos anteriores para identificar similitudes y diferencias entre ellos. Las ideas reunidas por estos análisis se pueden formular como conocimiento o los paquetes de experiencia pueden ser cualitativos, cuantitativos o una combinación de ambos. Ejemplos de paquetes cualitativos son patrones, heurísticas y mejores prácticas basadas en una serie de experiencias de diferentes fuentes. Ejemplos de cuantitativos son los modelos de paquetes de estimación basados en los atributos medidos de los proyectos y sus resultados presupuestados y reales. Otros ejemplos son el conocimiento que está empaquetado en términos de programas de software ejecutables que automatizan pasos del proceso de desarrollo basado en el conocimiento derivado de proyectos anteriores. Los estándares y recomendaciones de la industria también entran en esta categoría.

Primer nivel de la gestión del conocimiento

Soporte de gestión del conocimiento para actividades de ingeniería de software principal

Esta sección trata los procesos y actividades de ingeniería de software principal. (Birk, Surmann, & Althoff, 1999) ilustra El amplio espectro de procesos de ingeniería de software que pueden ocurrir en un típica proyecto de ingeniería de software. Lo que es común entre los resultados de todos estos procesos y actividades es que son todos documentos (incluso la fuente el código y los programas ejecutables pueden considerarse como documentos). El trabajo es, muchas veces, enfocado en la creación, revisión, edición y uso de estos documentos. Debido al hecho de que muchas organizaciones de software se distribuyen en grandes áreas geográficas, estos documentos deben estar disponibles de forma remota. Porque la ingeniería de software está tan dominada por los documentos que se producen durante las diversas actividades y procesos, que la base para un sistema de gestión del conocimiento es un sistema de gestión de documentos. De la mano con la gestión de documentos viene la necesidad de distribuir información sobre el proyecto, que exige la gestión de información general. La gestión de informacion se puede realizar utilizando herramientas de automatización de oficina para correo electrónico, gestión de tareas y programación. Un ejemplo de tal sistema es Microsoft Outlook en Combinación con Microsoft Exchange Server. Los sistemas de información general, sin embargo, caen fuera del alcance de este informe y no se discuten más. Los sistemas de gestión de documentos se han utilizado durante bastante tiempo, pero como el término la gestión del conocimiento se hizo popular, hubo una tendencia a volver a etiquetar el documento como herramientas de gestión del conocimiento, para adaptarse a la nueva tendencia.

Segundo nivel de gestión del conocimiento

Memoria organizacional para el desarrollo de software

Aprender de la experiencia requiere recordar la historia. La memoria individual, sin embargo, no es suficiente y toda la organización necesita una memoria para grabar explícitamente eventos críticos. Hay al menos tres formas distinguibles de memoria organizacional:

1. Memoria consta de documentos de trabajo regulares y otros artefactos que eran desarrollados principalmente para ayudar al desarrollo del producto (ejemplos en este categoría son especificación de requisitos y especificación de diseño)
2. Memoria formada por entidades que se desarrollaron específicamente para apoyar la memoria organizacional (ejemplos son lecciones aprendidas y análisis post mortem)
3. Una mezcla de las dos primeras formas.

En esta sección, describiremos las herramientas de software habituales utilizadas por los ingenieros de software en sus trabajos diario que también apoya la creación de una memoria organizativa. A veces la memoria organizativa se crea deliberadamente como parte de la herramienta. Otras veces es un efecto secundario deseable.

Una memoria organizacional para ingenieros de software respondería las preguntas ¿Por qué? ¿Quien? ¿Cuando? ¿Dónde? ¿Cómo?

Design Rationale es un marco para crear una memoria organizativa que intenta preservar la información sobre el desarrollo de un producto de software para que las preguntas Lo anterior puede ser respondido. La idea detrás de Design Rationale es que durante un proyecto de desarrollo de software se prueban muchas soluciones diferentes a los problemas y muchas decisiones. (Garcia, Howard, & Stefik, 1997)

Se realizan en base a los resultados de estas pruebas. El problema, sin embargo, es el razonamiento porque estas decisiones rara vez se capturan, lo que hace que sea muy difícil para alguien que no estaba involucrado en las decisiones para entender por qué el producto de software está diseñado de la forma en que esta. La lógica de diseño captura exactamente esta información. Razón de diseño también reconoce la necesidad de capturar información sobre soluciones que se consideraron, pero no implementado. La razón es que estos ejemplos "negativos" también podrían tener valor para un futuro mantenedor del sistema de software.

Mantenedores que no conocen la historia del diseño podría verse tentado a comenzar a rediseñar el sistema e incluso podría intentar Implementar soluciones que fueron rechazadas anteriormente por buenas razones. Si esas decisiones y sas razones detrás de ellos, argumenta Design Rationale, son capturadas en su lugar, luego futuros mantenedores tendran una mejor oportunidad de entender la historia y tomar mejores decisiones respecto a la evolución del sistema software. Un ejemplo de enfoque del design rationale se describe en (Potts & Bruns, 1988).

Si bien Design Rationale es un enfoque explícito para crear una memoria organizativa, herramientas regulares para el control de versiones, como el sistema de control de código fuente (SCCS) (Rochkind, 1975) representa una clase de herramientas que crean indirectamente tal memoria. Versión Control (VC) se utiliza para realizar un seguimiento de las diferentes versiones de un determinado documento. Versión control se puede aplicar a cualquier tipo de documento y es parte de muchos documentos en sistemas de gestión. Sin embargo, sus orígenes se remontan a la gestión de versiones de código fuente. Un sistema típico de control de versiones permite a los usuarios leer cualquier documento gestionado por el sistema. Para cambiar un

documento, el documento primero debe ser controlado. Una vez realizados los cambios, el documento vuelve a registrarse. Como parte del procedimiento de registro, el usuario comenta los cambios realizados. Mientras los cambios reales pueden ser rastreados usando programas que comparan dos documentos y muestran las diferencias entre ellos, el comentario debe indicar las razones detrás de los cambios cada versión de cada documento tiene un registro adjunto con información sobre quién hizo el cambio, cuándo se realizó, junto con el comentario que indica por qué se hizo el cambio. La información de diferencias indica lo que realmente se cambió. El uso regular de un sistema de control de versiones puede permitir a múltiples ingenieros de software trabajar casi simultáneamente en los mismos documentos y código fuente y puede permitir al Ingeniero de software para recuperar la última versión del sistema. Los registros de un VC también pueden ser utilizados como una memoria organizativa que indica cómo evolucionó el producto de software durante un cierto tiempo. También proporciona información sobre el proceso detrás de esta evolución. El ingeniero de software habitual puede utilizar esta información para identificar quién realizó un cambio determinado con el fin de encontrar un "experto". La información también se ha utilizado para el análisis avanzado de productos de software, así como procesos de software (Pena-Mora, 2006).

Mientras que Control de versiones ayuda a administrar versiones de archivos individuales, la gestión de configuración ayuda a gestionar composiciones de archivos. Esto es crucial para los ingenieros de software porque los productos de software consisten en conjuntos de archivos y cada compilación consta de diferentes versiones de estos archivos. En muchos casos, diferentes clientes utilizan diferentes compilaciones de sistema de software. Por lo tanto, existe una gran necesidad de que las organizaciones realicen un seguimiento de exactamente qué

versión de cada archivo de código fuente entró en una compilación particular que un determinado cliente tiene problemas. Sin tener esta información en la memoria organizacional, sería imposible recrear y detectar los problemas reportados por los clientes para su propia versión del sistema de software. Los requisitos de software impulsan el desarrollo de sistemas de software, pero hay muchos que afirma que la conexión entre el sistema final y sus requisitos es confusa (Soloway, 1997). Esto crea varios problemas. Primero, algunos contratos especifican que el proveedor debe poder relacionar cada pieza de código con uno o más requisitos. Segundo, los cambios y adiciones al producto a menudo se formulan en términos de nuevos requisitos, por lo que es necesario evaluar el impacto de los requisitos nuevos y modificados en el producto con el fin de determinar el costo. La trazabilidad es un enfoque que hace la conexión entre los requisitos y el sistema de software final explícito. Los requisitos de rastreo contribuyen a la memoria organizacional y ayudan a responder. Preguntas como "¿Qué requisitos llevaron a una pieza particular de código fuente?" y "¿Qué código fue desarrollado para satisfacer este requisito en particular?

Tercer nivel de gestión del conocimiento:

Conocimiento empaquetado que soporta la aplicación de conocimiento

Hay una gran cantidad de herramientas disponibles, ya sea como prototipos de investigación o como herramientas comerciales, que afirman estar basadas en el conocimiento. Común para estas herramientas es que están específicamente diseñados para la ingeniería de software. Ellos apoyan al ingeniero de software en su trabajo diario y a menudo resulta del análisis de conocimiento de muchos proyectos anteriores. Se han realizado encuestas y consultorias sobre herramientas que ofrecen apoyo adecuado a la gestión del conocimiento en

ingeniería de software. Estas encuestas, por necesidad, se basan en las afirmaciones hechas por desarrolladores, vendedores y usuarios. Estas herramientas se han clasificado de 2 maneras: Primero, De acuerdo a la tarea que realizan en el ciclo de ingeniería del software. Segundo, según el ciclo de gestión del conocimiento. Clasificacion según el ciclo de gestión del conocimiento resultó en tres categorías principales

1. Herramientas que soportan el despliegue y aplicación del conocimiento.

2. Herramientas de apoyo a la adquisición de conocimientos.

3. Herramientas de organización del conocimiento.

Conclusiones.

El enfoque de este informe es la gestión del conocimiento en ingeniería de software. Presenta los desarrollos en la gestión del conocimiento en general, y para la ingeniería de software en particular, y discute modelos, enfoques y herramientas para la gestión del conocimiento. El informe también presenta recursos que pueden proporcionar ayuda, inspiración e información para organizaciones que quieran gestionar mejor sus conocimientos. El desarrollo de software es una actividad intensiva en conocimiento y personas. Los grupos que son distribuidos geográficamente realizan una importante cantidad de trabajo en ingeniería de software. Las personas en tales grupos deben colaborar, comunicarse y coordinar sus trabajos, que hace de la gestión del conocimiento una necesidad. Como cuestión de hecho, las pequeñas organizaciones estables donde los empleados están al alcance de la mano unos de otros pueden probablemente sobreviva sin gestión del conocimiento. Sin embargo, para las grandes

organizaciones, cuyo entorno está cambiando continuamente, o tienen una alta rotación, la gestión de sus activos de conocimiento es fundamental para la supervivencia.

Una característica de la ingeniería de software que resulta ser una ventaja sobre otras industrias en términos de gestión de capital intelectual es que los artefactos ya se capturan en forma electrónica y se pueden almacenar y compartir fácilmente. Además, los ingenieros de software a menudo tienen una actitud amistosa hacia el uso de nuevas tecnologías. Esto significa que una organización que implemente un sistema de gestión del conocimiento podría tener una buena oportunidad para tener éxito con esta misión. Sin embargo, esto sigue siendo una tarea desafiante porque un sistema de gestión del conocimiento es más que solo tecnología. Sólo hay unos pocos Informes publicados sobre iniciativas para gestionar el conocimiento en organizaciones de software, pero todos de ellos hablan de la dificultad de lograr la aceptación de los empleados y la implementación del sistema de gestión del conocimiento de una manera que maximiza la ayuda proporcionada a sus usuarios. Una organización desarrolladora de software que busca implementar la gestión del conocimiento puede encontrar estos informes muy útil.

Bibliografía.

Bennis, W., & Biederman, P. (1998). *Ninguno de nosotros es tan inteligente como todos nosotros.*

Birk, A., Surmann, D., & Althoff, K. (1999). *Applications of Knowledge Acquisition in Experimental Software Engineering.*

Garcia, A., Howard, H., & Stefik, M. (1997). *Documentos de diseño activo: un nuevo enfoque para la documentación de apoyo en el diseño de rutina preliminar.*

Pena-Mora, F. (2006). *Aumento de patrones de diseño con fundamentos de diseño, inteligencia artificial para el diseño, análisis y fabricación de ingeniería.*

Potts, C., & Bruns, G. (1988). *Recording the Reasons for Design Decisions.*

Rochkind, M. (1975). *The Source Code Control System.*

Soloway, E. (1997). *I Can't Tell What in the Code Implements What in the Specs.*