Gestión del Conocimiento
en
Enfoques de Desarrollo de Software Tradicional y Agilista

Broderick Crawford Labrin
Escuela de Ingeniería Informática, Facultad de Ingeniería
Pontificia Universidad Católica de Valparaíso (PUCV)
Av. Brasil 2241, Valparaíso, Chile
Fono 32-273761
broderick.crawford@ucv.cl

Resumen

A la luz de los conceptos básicos de la Gestión del Conocimiento se caracterizan las prácticas esenciales de los enfoques tradicional y agilista de desarrollo de software. Los métodos de desarrollo ágil comparten valores y principios publicados en el Manifiesto Agilista, promoviendo a los individuos y sus interacciones por sobre los procesos y herramientas. Asumiendo las características de cambio acelerado e incertidumbre del actual entorno, enfatizando el tratamiento del conocimiento tácito sobre el explícito reemplazando la generación de documentación detallada por la comunicación cara a cara. Los métodos tradicionales en cambio, usan fuertemente la documentación para capturar conocimiento que se obtiene en cada una de las actividades del ciclo de vida de un proyecto. El crear y compartir conocimiento son relevantes para ambos enfoques de desarrollo de software, por lo que resulta interesante abordarlos desde el punto de vista de la Gestión del Conocimiento, en el presente trabajo se incluyen sus principales conceptos y la relación con los enfoques de desarrollo tradicional y ágil, enfatizando el análisis en una de las actividades más relevantes del ciclo de vida: la elicitación de requerimientos.

1. Introducción

La Ingeniería de Software es un proceso intensivo de conocimiento en cada una de sus etapas, y dado que en ningún proyecto todos los involucrados tienen todo el conocimiento requerido, resulta claro que existirán diversas actividades donde se deberá crear, transferir y difundir conocimiento. Históricamente, los enfoques de desarrollo han facilitado el compartir conocimiento a través de la documentación. La que permite hacer visible el proceso de desarrollo de software a través de la abundante cantidad de documentos de especificación que se producen en cada fase. Actualmente, se puede observar un amplio espectro de diferentes formas de abordar un proceso de desarrollo de software, en cuyos extremos destacan por un lado los conservadores y formalistas, quienes se apegan a un proceso de desarrollo tradicional bien planificado, haciendo énfasis en los detalles para asegurar la calidad de sus productos, aspirando a niveles de control y repetición. Y en el otro extremo, el manifiesto agilista [1] que valora más las respuestas a los cambios, a través del uso de fuertes interacciones con los usuarios y entre los desarrolladores, que el seguimiento estricto a una planificación. Argumentan que la rigidez de los planes o contratos muchas veces deriva en que cuando el sistema es entregado el problema que se pretendía resolver cambió o ya no existe. El crear y compartir conocimiento son relevantes para ambos enfoques de desarrollo de software, por lo que es interesante abordarlos desde el punto de vista de la Gestión del Conocimiento (GC). En el presente trabajo se incluyen los principales conceptos de la GC y su relación con los enfoques de desarrollo Tayloriano y Agilista enfatizando el análisis en una de las actividades más relevantes del ciclo de vida: la elicitación y especificación de requerimientos.

2. Marco General de Gestión del Conocimiento

En la actualidad nos encontramos en una era denominada la "Sociedad del Conocimiento" caracterizada principalmente por la aparición continua de nuevo saber, con amplias posibilidades de divulgación y donde la empresa pasa a ser concebida desde la idea de un "conjunto de activos tangibles organizados en un determinado proceso productivo para lograr unos objetivos concretos", hasta su consideración actual como "conjunto de activos intangibles, generadores de un capital intangible o capital intelectual"[5]. Con la finalidad de poder aprovechar y obtener mayores ventajas del conocimiento organizacional es que nace la GC, en adelante se entenderá por gestión "el proceso por el cual se obtiene, despliega o utiliza una variedad de recursos básicos (en este caso recursos intangibles como el conocimiento) para apoyar los objetivos de la organización"[6]. La GC es un proceso a través del cual las organizaciones logran descubrir, utilizar y mantener el conocimiento que poseen, con la idea de alinearlo con las estrategias de negocio para la obtención de ventajas competitivas.

En GC son reconocidos los modelos de Nonaka y Takeuchi [7], y el modelo Intelect [8], los cuales plantean que las personas juegan un rol importante y determinante en la generación de valor de las organizaciones, debiendo estas últimas crear las estructuras necesarias para la generación de ventajas competitivas. Es así, que la gestión de los recursos intangibles tiene como objetivo convertir el capital humano (habilidades, conocimientos, etc.) en capital estructural (conocimiento organizacional o lo que queda cuando los empleados se van a sus casas), reduciendo el riesgo de perder valioso conocimiento si la gente deja la organización. De esta, manera lo importante no es retener a las personas, sino que mantener y mejorar los sistemas de manera de poder continuar generando conocimiento.

La Ingeniería de Software es un proceso intensivo de conocimiento, donde interactúan usuarios y desarrolladores. El usuario brinda una concepción de la funcionalidad esperada y el desarrollador especifica esta funcionalidad a partir de esta primera concepción mediante aproximaciones sucesivas. Este ambiente de interacción motiva la búsqueda de estrategias robustas para garantizar que los requisitos del usuario sean descubiertos con precisión y que además sean expresados en una forma correcta y sin ambigüedad, y que sea verificable, trazable y modificable. El crear y compartir conocimiento son relevantes para cualquier enfoque de desarrollo de software, los enfoques de desarrollo tradicionales han facilitado el compartir conocimiento a través de documentación intensiva, pero actualmente se puede observar numerosos métodos que valoran más el uso de fuertes interacciones con usuarios.

Los métodos, herramientas y las actuales implementaciones de GC en muchas compañías han seguido también estos dos caminos: documentación intensiva e interacciones. En GC estos enfoques se han denominado Enfoque Centrado en el Producto y Enfoque Centrado en el Proceso. El Enfoque Centrado en el Producto, tal como el enfoque de desarrollo tradicional, se basa en la documentación del conocimiento: creación de documentos, repositorios y posibilidad de reutilización. También se conoce como enfoque centrado en el contenido o "codificación". Considera que el conocimiento es una "cosa" que puede ser manipulada como un objeto independiente. Que es posible capturar, distribuir, medir y gestionar. El Enfoque Centrado en el Proceso, al igual que los Métodos Agiles, principalmente considera la GC como un proceso de comunicación y colaboración. Donde el conocimiento está estrechamente relacionado con la persona que lo desarrolla y es compartido con otros a través de contactos personales. También se le conoce como enfoque de colaboración o "personalización". Pone énfasis en las formas de promover, motivar, estimular o guiar el proceso de aprendizaje, descartando la idea de tratar de capturar y distribuir conocimiento [9]. La elección del enfoque más apropiado, o los intentos de combinarlos, dependerán de las características del proyecto, las personas y la organización [10].

3. Las metodologías Ágiles versus las Tradicionales

Cuando se compara métodos ágiles y tradicionales, hay autores que sugieren utilizar el término *enfoques Taylorianos* para referirse a las metodologías tradicionales [3, 4]. Argumentando que otros términos también utilizados para referenciarlos, como *métodos planificados o disciplinados*, no son los más adecuados. Ya que los métodos ágiles también son disciplinados e involucran tareas de planificación, la diferencia radica en el alcance de esta planificación, no en su inexistencia. Las metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo de software, con el objetivo de asegurar que el producto que se obtenga satisfaga los requerimientos del usuario y reúna estándares aceptables de calidad. El trabajo de planificación es riguroso, aún cuando en la práctica muchas veces estas planificaciones no se respetan. En contraposición, las metodologías ágiles aportan nuevos métodos de trabajo que apuestan por una cantidad apropiada de procesos. Es decir, no se desgastan con una excesiva cantidad de cuestiones administrativas (planificación, control, documentación) ni tampoco defienden la postura extremista de total falta de proceso. Ya que se tiene conciencia de que se producirán cambios, lo que se pretende es reducir el costo de rehacer el trabajo [1,11].

Las metodologías ágiles son adaptativas más que predictivas. Una metodología tradicional potencia la planificación detallada y de largo alcance de prácticamente todo el desarrollo de software (ejemplo Modelo Cascada). En contraste, las metodologías ágiles proponen procesos que se adaptan y progresan con el cambio, llegando incluso hasta el punto de cambiar ellos mismos. Las metodologías ágiles están orientadas más a los desarrolladores que a los procesos de desarrollo. Intentan entonces trabajar con la naturaleza de las personas (desarrolladores y usuarios) asignadas a un proyecto, más que contra ellos, de tal forma que permiten que las actividades de desarrollo de software se conviertan en una actividad de colaboración grata e interesante [12]. Se presentan algunas limitaciones a la aplicación de los procesos ágiles en algunos tipos de proyectos. Los agilistas y sus críticos focalizan la discusión en torno a temas como la documentación y codificación. Por un lado se sostiene que el código es el único entregable que realmente importa, desplazando el rol del análisis y diseño en la creación de software. Los críticos precisan que el énfasis en el código puede conducir a la pérdida de la memoria corporativa o conocimiento organizacional, porque hay poca documentación y modelos para apoyar la creación y evolución de sistemas complejos. [13].

4 Relevancia de la Elicitación de Requerimientos

Ninguna otra actividad del proceso creativo de desarrollo es tan vital como el establecimiento detallado de sus requerimientos, tan significativa en sus resultados finales y tan difícil de corregir. La inestabilidad de los requerimientos se puede atribuir a que los usuarios no saben lo que quieren, la explicación puede surgir atendiendo un tipo de usuario business man, cuyo mindset es: “give me what I say I want, then I can tell you what I really want” [14]. Lo que parece confirmar que las necesidades del usuario y su solución muchas veces son tácitas y difíciles de explicitar. Como en muchos casos es imposible especificar completamente, precisamente y correctamente los requerimientos del producto de software antes de desarrollar alguna versión, desde hace ya muchos años la esencia de este problema se ha tratado de enfrentar haciendo uso de prototipos. Considerando que la mayoría de los usuarios se relacionan mejor con un prototipo de pantalla que leyendo un documento de requerimientos. Los agilistas plantean que el software funcionando genera mejor y más rápido conocimiento, por lo cual privilegian el uso de iteraciones muy cortas y la refactorización. En cambio, el modelo de desarrollo tradicional no proporciona este feedback, por lo que también se le considera un modelo determinístico, al suponer que los detalles del proyecto pueden ser completamente determinados desde su inicio. Los enfoques ágiles consideran la

evolución de los requerimientos a través de todo el ciclo de vida, utilizan procesos técnicos y de gestión cuyo objetivo es adaptarse continuamente a los cambios que surgen del conocimiento generado en conjunto con el usuario durante el proceso de desarrollo. Los seguidores de los métodos más formales no pueden entender cómo se podría construir algo que trabaje bien sin analizar los requerimientos. Pero los agilistas no se conforman con perder tiempo valioso y escaso en analizar requerimientos que probablemente cambiarán [2,15]. Hay un aspecto común relevante en los agilistas: las iteraciones. Una iteración es un incremento de software que es diseñado, programado, testeado, integrado y liberado durante un corto período de tiempo. Esta forma de trabajar aumenta considerablemente el feedback, comunicación y colaboración, entre usuarios y desarrolladores. El testing es incluído tempranamente, los ambientes de hardware y software son probados desde el comienzo. Y todos los problemas en general son descubiertos a tiempo con este enfoque tolerante a los cambios de requerimientos [16].

5. Tratamiento de la Documentación

Los enfoques Taylorianos externalizan el conocimiento en un gran número de documentos cuyo objetivo principal es darle visibilidad al proceso de desarrollo y asegurar que los requerimientos, el diseño, la construcción, la mantención y también la gestión del proyecto sean bien entendidos y ejecutados por los involucrados. Una ventaja de la externalización de conocimiento a través de la documentación es que se reducen las posibilidades de pérdida de conocimiento como resultado del personal que abandona el proyecto y se mejoran las posibilidades de atender proyectos futuros. También permite la colaboración de los equipos de trabajo distribuidos temporal o físicamente.

Por otro lado, hay una gran "cantidad" de conocimiento que es tácito, y que es difícil hacerlo explícito, y muchas veces poco de este conocimiento explícito puede ser documentado en detalle debido a que los equipos de desarrollo perciben esta tarea como larga y tediosa, sin lograr apreciar sus beneficios. Aún, si fuera posible documentar este conocimiento, queda pendiente la manera de asegurar su actualización. Así, los métodos ágiles promueven sólo la generación y uso de documentación necesaria y significativa, que mejore la comunicación o entendimiento del sistema. Con modelos no muy detallados y de uso público para todo el equipo de desarrollo, facilitando la difusión del conocimiento. Para compensar la reducción de conocimiento explícito y documentación, los métodos ágiles practican la permanente socialización (comunicación y colaboración) de los desarrolladores y usuarios apuntando a enriquecer el conocimiento tácito. Lo que funciona en equipos pequeños y generalmente colocalizados. En equipos dispersos geográficamente o de muchas personas, la comunicación cara a cara no funciona, y la documentación se hace necesaria para poder compartir conocimiento. Existen estudios sobre el uso de métodos ágiles en ambientes de trabajo distribuido que muestran cómo el uso de herramientas para GC han posibilitado levantar en parte esta restricción [18,19].

6. Ventajas de la Co-localización del Equipo

En relación a la elicitación de requerimientos, los métodos ágiles propugnan la activa participación de los usuarios en el mismo lugar de trabajo y por períodos largos de tiempo. Respecto a las prácticas utilizadas, éstas no difieren significativamente de las utilizadas en algunos proyectos tradicionales (Entrevistas, Casos de Uso, Storyboard, Brainstorming, Joint Application Design, Focus Groups [17]), la diferencia más significativa radica en que los métodos Taylorianos no promueven las prácticas que aseguren la participación del usuario. En este aspecto cabe recordar que algunos de los principios del movimiento agilista son: "los usuarios y desarrolladores deben trabajar juntos diariamente durante el proyecto" y que "el método más eficiente y eficaz de compartir la información es la conversación cara a cara". A través de la colaboración y el diálogo permanente de los participantes, sus modelos mentales y experiencia son compartidos. Es la

socialización que plantea Nonaka, que se fundamenta en esta posibilidad de poder compartir el tiempo y espacio haciendo colectivo el conocimiento tácito.
En los enfoques tradicionales, donde los requerimientos deben ser especificados previos al desarrollo, el equipo a cargo de tareas de diseño y construcción interactúa poco o nada con los usuarios con el objetivo de entender los requerimientos del sistema. Basando su trabajo fundamentalmente en los documentos de especificación. Este enfoque es adecuado sólo para sistemas cuyos requerimientos se sabe con certeza permanecerán estables. Pero en los escenarios actuales de rápido y constante cambio los requerimientos son inestables

7. Los Individuos y el Ambiente de Trabajo

El manifiesto agilista dice: "Construya los proyectos alrededor de individuos motivados. Déles el ambiente y apóyelos en lo que necesiten. Y confíe en ellos". Lo que hace mucho sentido dada la naturaleza social del proceso de desarrollo. Es importante desarrollar las confianzas necesarias entre los miembros, usuarios y desarrolladores. Lo que facilitará la creación, compartición y difusión del conocimiento [21]. La propiedad colectiva del código, reuniones periódicas, usuarios co-localizados, programación en parejas, son prácticas que requieren de confianza para poder ejecutarse, y que a su vez promueven la confianza. Un factor relevante en la socialización, tiene relación con los contactos y colaboraciones voluntarias de los miembros del equipo. En el enfoque agilista se considera equipos de trabajo polifuncionales, en cambio los proyectos tradicionales trabajan con equipos subdivididos en roles. Los roles permiten subdividir las tareas, lo que puede facilitar la asignación a diferentes proyectos que se desarrollan en paralelo. Así, se podrá optimizar el uso del recurso "analista", "diseñador" o "programador" a similitud de un proceso de fabricación. Una desventaja de este enfoque es la falta de comunicación y flujos de información entre los diferentes roles de un equipo asignado a un proyecto, lo que dificulta la socialización (el compartir conocimiento tácito), perpetuando el conocimiento aislado de cada miembro.

8. Conclusiones

A la luz de lo planteado por Nonaka, los enfoques tradicionales no consideran instrumentos que permitan abordar cómo los involucrados internalizan conocimiento explícito o puedan compartir conocimiento tácito que no es externalizado. Los métodos ágiles se fundamentan en la socialización, por medio de la comunicación cara a cara, la colaboración, y la programación en parejas, para compartir conocimiento tácito entre los desarrolladores y usuarios. Usando la externalización e internalización para transferir conocimiento. El enfoque iterativo de desarrollo utilizado posibilita una buena retroalimentación y un aprendizaje continuo entre los usuarios y desarrolladores. Para facilitar el aprendizaje entre los desarrolladores, los métodos ágiles hacen uso de diversas prácticas, entre las que se destacan: reuniones periódicas (diarias o semanales), programación en parejas, propiedad colectiva del software, refactorización. Los métodos ágiles privilegian las personas, la comunicación y la colaboración. Lo que facilita el compartir conocimiento tácito. Fomentan también, una cultura de confianza y motivación, lo que a su vez posibilita la implementación de herramientas para compartir conocimiento explícito.
Ciertos dominios pueden ser más adecuados a un tipo de proceso. Y en general, algunos aspectos del desarrollo de software se beneficiarán del enfoque agilista mientras otros obtendrán beneficios de un enfoque tradicional (predictivo, menos ágil o Tayloriano). Una combinación de experimentación, revisión e iteración en muchos casos puede presentar buenos resultados. El objetivo debe ser balancear experimentación y revisión. Este trade off debe ser resuelto en base a los costos involucrados en la generación del conocimiento. Si el costo de probar es muy alto, es preferible generar conocimiento a través de la deliberación y revisión. En otros casos, la experimentación puede ser menos costosa y permitir generar conocimiento rápidamente.

Muchos autores a la fecha vienen indicando que las metodologías tradicionales no son totalmente adecuadas para todos los desarrollos software. Las razones son diversas, las principales son la falta de flexibilidad de los procesos de desarrollo frente a cambios y su excesiva documentación. Como se ha planteado en este trabajo los métodos ágiles y los tradicionales no son estrictamente competidores directos. Cada uno de ellos tiene su propio segmento de aplicación o terreno. Y pueden ser usados en proyectos con diferentes características: los métodos tradicionales son más adecuados en grandes proyectos con requerimientos estables, aplicaciones críticas y donde la memoria corporativa tiene relevancia. Los métodos ágiles en cambio se adecuan mejor en ambientes dinámicos, con equipos de trabajo pequeños y produciendo aplicaciones no críticas. También son una buena elección cuando se trabaja con requerimientos desconocidos o inestables, garantizando un menor riesgo ante la posibilidad de cambio en los requerimientos.

6. Referencias.

[1] K. Beck et al, *Manifesto for Agile Software Development*, http://agilemanifesto.org/
[2] J. Ridderstrale & K. Nordstrom, *Funky Business: Talent Makes Capital Dance*, Financial Times / Prentice Hall, EEUU, 2000
[3] T. Chau y F. Maurer, *Knowledge Sharing un Agile Software Teams*, http://sern.ucalgary.ca/~milos/papers/2004/ChauMaurer2004.pdf/
[4] F. Taylor, *The Principles of Scientific Management*, Dover Pubns, 1998
[5] E. Bueno, *La Gestión del Conocimiento nuevos perfiles profesionales*, Junio 1999. http://www.sedic.es/bueno.pdf
[6] H. Koonz y H. Weihrich, *Administración: una perspectiva global*, McGraw Hill, España, 1995
[7] S. Nonaka y N. Takeuchi, *La Organización creadora de Conocimiento*, Oxford University, 1995
[8] Instituto Universitario Euroforum Escorial, *Medición del Capital Intelectual: Modelo Intelect*, Madrid, 1998
[9] G. Mentzas, *The two faces of Knowledge Management*, http://imu.iccs.ntua.gr/Papers/O37-icg.pdf/
[10] D. Apostolou y G. Mentzas, *Experiences from KM implementations in companies of the software sector*, Business Process Management, Vol 9 No 3, 2003, MCB UP Limited
[11] U. Kelter et al, *Do we Need 'Agile' Software Development Tools?*, http://pi.informatik.uni-siegen.de/
[12] M. Fowler, *The New Methodology*, http://www.programacionextrema.org/articulos/newMethodology.html/
[13] J. Bozo y B. Crawford, *Métodos Ágiles como Alternativa al Proceso de Desarrollo Web*, IX Congreso Argentino de Ciencias de la Computación CACIC 2003, La Plata, del 6 al 10 de Octubre 2003
[14] W. H. Inmon, *Choosing the right approach to DW: Big Bang vs. Iterative*, www.billinmon.com
[15] B. Meyer, *Software Enginnering on Internet Time*, IEEE Computer, Vol. 34, Nº35, May 2001
[16] M. Poppendieck y T. Poppendieck, *Lean Development Thinking Tools for Agile Software Development Leaders*, Addison-Wesley Pub Co, 2003
[17] M. J. Escalona y N. Koch, *Ingeniería de Requisitos en Aplicaciones para la Web: Un estudio comparativo*, www.pst.informatik.uni-muenchen.de/personen/kochn/ideas03-escalona-koch.pdf
[18] P. Baheti, L. Williams, E. Gehringer, y D.Stotts, *Exploring Pair Programming in Distributed Object-Oriented Team Projects*, In Proceedings of XP/Agile Universe 2002, Chicago, August 4-7, 2002, Lecture Notes in Computer Science 2418, Springer Verlag
[19] I. Rus, M. Lindvall, *Knowledge Management in Software Engineering*, IEEE Software, May-June 2002
[20] B. Crawford, J. Bozo y K. Rojas, *Marco teórico para la proposición fundamentada de una herramienta computacional para la Gestión de Competencias*, XI Encuentro Chileno de Computación, Jornadas Chilenas de Computación 2003, Chillán, del 3 al 7 de Noviembre 2003
[21] A. Cockburn y J. Highsmith, *Agile Software Development: The People Factor*, IEEE Computer, Vol. 34, No.11, 2001