Inserción del mantenimiento en los procesos ágiles

Karla Mendes Calo[1] Karina Cenci[1] Pablo Fillottrani[1,2]

[1]Laboratorio de Investigación en Ingeniería de Software y Sistemas de Información
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
[2]Comisión de Investigaciones Científicas, Provincia de Buenos Aires
{kmca,kmc,prf}@cs.uns.edu.ar

Resumen El mantenimiento del software abarca todas las actividades asociadas con el proceso de cambio del software. Los artefactos generados durante la etapa de desarrollo, son utilizados como soporte en el mantenimiento de las aplicaciones. Las metodologías ágiles valoran y promueven la comunicación verbal entre los integrantes del equipo por sobre la documentación. Frecuentemente esto origina errores o demoras involuntarias en el mantenimiento, en especial, si decisiones relevantes no están registradas.
En este trabajo, se expone una práctica no ágil, que se adapta al modelo ágil en la etapa de mantenimiento, la cual consiste en una Iteración de Reordenamiento del Conocimiento (IRC) que se lleva a cabo antes de iniciar el proceso de mantenimiento ágil, recomendando la utilización de esta práctica en proyectos que superan el año de desarrollo, y se utilizó una metodología ágil para el proceso. Se muestra cómo la combinación y adaptación de ciertas prácticas utilizadas en metodologías tradicionales con metodologías ágiles, permite obtener procesos híbridos, con los beneficios de ambas metodologías.

Palabras claves: Mantenimiento - Procesos Ágiles - Gestión Conocimiento

1. Introducción

Tradicionalmente, el proceso de desarrollo de software ha sido descripto como un proceso secuencial. Un modelo secuencial extensamente conocido es el modelo cascada, en la que una fase puede ser iniciada solo si todas las fases precedentes fueron completadas. Como contraste de este proceso, las metodologías ágiles se caracterizan por realizar entregas de software al finalizar cada iteración, cuya duración típicamente no excede el mes. En cada iteración se incluyen todas las fases del desarrollo de software.

El término desarrollo de software ágil fue introducido en 2001 por el Manifiesto Agil [3], y desde entonces, se han publicado varios artículos sobre el tema. Esta metodología enfatiza la simplicidad y la comunicación, tanto entre los miembros del equipo y con el cliente, minimizando la documentación formal. Cambios en los requerimientos durante las iteraciones no solo son esperados, sino que son fomentados, posibilitando cambiar de

dirección rápidamente de acuerdo a las necesidades del cliente, impulsando el concepto de agilidad.

La información, el conocimiento y la gestión del conocimiento son fundamentales para el desarrollo de cualquier proyecto de software. Para el caso de metodologías ágiles, la información se puede encontrar en distintos tipos de documentos: e-mails entre el cliente y miembros del equipo, posteos acerca de discusiones técnicas con alternativas para su resolución, lecciones aprendidas, historias gestionadas por un grupo de procesos centrales. Dado que el proceso de construcción de software utilizando metodologías ágiles es un proceso cíclico e iterativo, hay que tener en cuenta que la gestión de este conocimiento, implica asegurar la actualización y distribución del conocimiento entre los miembros del equipo, y combinarlo además con nuevo conocimiento que se irá sumando a través de las iteraciones durante el proceso de desarrollo.

Si bien la entrega del producto completo es el hito más importante al que un equipo de desarrollo aspira, las tareas en el proceso de desarrollo de software no terminan allí. Inevitablemente sufren cambios para su permanencia y utilidad. Aparecen nuevos requerimientos por parte de los usuarios que involucrará el desarrollo de nuevas funcionalidades, o bien por cambios en normativas legales e impositivas, cambios tecnológicos, errores detectados en su funcionamiento, mejoras en la performance, mejoras en el rendimiento. Estos cambios en algunos casos pueden involucrar reingeniería de alguna parte del producto. A esta etapa del desarrollo de software, lo denominamos mantenimiento.

El resto del trabajo está organizado de la siguiente manera. En la sección 2, se introduce conceptos, políticas y tipos de mantenimiento. Sección 3, son presentadas las características del mantenimiento en las metodologías ágiles. Sección 4 presenta un caso de estudio y en sección 5 experiencias de mantenimiento en las metodologías ágiles. La propuesta de una metodología híbrida para el mantenimiento, se presenta en la sección 6, y por último conclusiones.

2. Mantenimiento

El mantenimiento del software no es como el mantenimiento del hardware, que es la devolución del artículo (ítem) a su estado original. El mantenimiento del software desplaza un artículo de su estado original. El mismo abarca todas las actividades asociadas con el proceso de cambio del software. Esto incluye todo lo asociado con correcciones de error (*bug*), mejoras funcionales y de rendimiento, proporcionar compatibilidad con versiones anteriores, actualización del algoritmo, la creación de métodos de acceso a la interfaz de usuario, y cualquier otro cambio.

Mantenimiento de Software es definido en el standard del IEEE para el Mantenimiento de Software, IEEE 1219, como la modificación de un producto de software después de entregado para corregir fallas/defectos, para mejorar el rendimiento u otros atributos, o para adaptar el producto a los cambios del ambiente.

En software, agregar una autopista de seis carriles de autos de un puente del ferrocarril es considerado mantenimiento, y es particularmente valioso si el mismo se puede realizar sin frenar el tráfico de los trenes. El desafío es el diseño de software de tal manera que el mantenimiento se pueda realizar sin frenar al software que está en producción.

Las políticas sobre el mantenimiento presentadas en [9] son las siguientes: a) *Tradicional* No considera la posibilidad de mantenimiento. b) *Nunca* Decide que nunca va a ocurrir mantenimiento. Simplemente se escriben muy buenos programas correctos

desde el inicio. c) *Discreto* En teoría, el proceso acepta el hecho del cambio, se mantienen listas de partes y herramientas sobre cada ítem, los cambios son realizados bajos estrictos controles. d) *Continuo* El cambio es constante. La migración de hardware, software y comportamiento durante el funcionamiento del sistema es necesaria.

En los desarrollos actuales de productos de software, se acepta el hecho del cambio, pero no siempre el diseño y arquitectura están preparados para soportar las diversas modificaciones sin comprometer las calidades (rendimiento, confiabilidad, etc) del mismo. En función de la clase de cambio que deba llevarse a cabo, se puede clasificar el mantenimiento como a) *Adaptativo* Modificaciones para adaptar el producto del software a los cambios en los requerimientos de datos y procesamiento del ambiente (entorno). [2], b) *Preventivo* Modificaciones al producto de software después de entregado para detectar y corregir fallas latentes antes de que se conviertan en fallas funcionales. [4], c) *Correctivo* Tiene como objetivo solventar una deficiencia en un componente del sistema de información (puede ser software o documental). Entiéndase deficiencia como algo que debería funcionar o estar correcto y que no lo está., d) *Perfectivo* Modificaciones al producto de software después de entregado para detectar y corregir fallas latentes en el producto de software antes de que se manifiesten como fallas. [4], e) *Ayuda al Usuario* Responde a las demandas de los usuarios distintas de las adaptativas, correctivas, preventivas o perfectivas. [2].

Existen algunos problemas que son específicos del mantenimiento, como es la necesidad de descubrir decisiones de diseño de alto nivel, alto volumen de información a considerar, necesidad de entrenamiento de nuevo personal, resistencia al cambio por desgaste de la arquitectura.

En las metodologías tradicionales, un aspecto significativo que gobierna el proceso es el exhaustivo uso de artefactos, para registrar el proceso y la documentación. La documentación, si es correcta, completa y consistente, ha sido considerada como una poderosa herramienta para los ingenieros de software para alcanzar el éxito. En contrapartida, una pobre documentación es considerada la principal razón para la rápida degradación de la calidad del software y el envejecimiento. El propósito de la documentación no es solamente describir el sistema de software sino también registrar el proceso. Los ingenieros de software necesitan poseer un buen conocimiento del sistema para estar habilitados a continuar con la evolución del mismo.

En las metodologías tradicionales, el mantenimiento se apoya en gran medida en los artefactos generados en etapas de análisis y diseño. Estos artefactos generados son el soporte fundamental del conocimiento para llevar a cabo el mantenimiento de las aplicaciones.

La figura 1 muestra cómo es el despliegue de un producto, considerando en el desarrollo todas las etapas necesarias para construir el producto de software, a partir de la entrega del mismo se pasa a la etapa de mantenimiento que se encarga de todos los tipos de modificaciones requeridas.

3. Mantenimiento de Software en las Metodologías Ágiles

La documentación en ambientes ágiles no está registrada formalmente, la transferencia es informal y primordialmente en forma oral. Se considera a la documentación como un aspecto secundario, centrando el proceso en el producto.

En ambientes ágiles, un riesgo significativo es que la mayoría de la documentación está en la memoria de los desarrolladores o en posters temporarios. Con la ausencia de una apropiada documentación, existe un alto riesgo que el conocimiento organizacional

Figura 1. Ciclo de Despliegue

de un sistema sea olvidado, mal interpretado o perdido. Una alta rotación del personal puede conducir a una pérdida significativa del conocimiento del sistema si un miembro habilidoso/dotado abandona y su conocimiento y experiencia no está registrada. Además, existen dificultades al momento de entrenar a nuevos contratados cuando la documentación se encuentra en el código.[5]

La figura 2 muestra el despliegue de un producto utilizando una metodología ágil, el desarrollo del mismo está gobernado por las iteraciones, donde cada una de las iteraciones genera un hito, que es un subconjunto de requerimientos desarrollados para poner a producción. A partir del momento que el producto está en producción pueden aparecer defectos en el mismo que hay que corregir, el modelo de proceso está gobernado por los cambios permanentes a través de nuevos requerimientos y de adaptaciones al producto en funcionamiento.

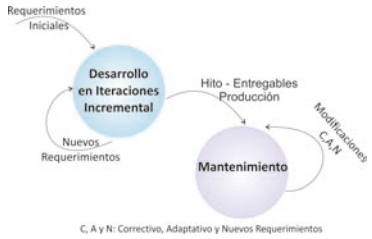

Figura 2. Ciclo de Despliegue

Las metodologías ágiles son adoptadas cada vez con más frecuencia no solo como ciclo de vida en la etapa de desarrollo, sino que también tienden a utilizarse en el proceso de mantenimiento, ya que el uso de distintos procesos en una organización generalmente disminuye la eficiencia del negocio [7] [10]. En función de esto, hay una necesidad de revisar el mantenimiento en un contexto ágil.

Svensson y Host [11] presentaron un caso de estudio de un mantenimiento de software, y los resultados mostraron que un gran número de prácticas ágiles tales como *planning game*, programación por pares y la integración continua son apropiados en el contexto del mantenimiento y facilitan la transición del conocimiento. Otros trabajos sobre el mismo aspecto son Shaw [8] y Rico [6].

La adopción de metodologías ágiles en la etapa de mantenimiento debe ser llevado a cabo con cierta cautela, debido justamente a los escasos artefactos generados durante las etapas de desarrollo, que sirvan de soporte en el mantenimiento de las aplicaciones.

Consideraremos que el alcance de las responsabilidades del equipo ágil en el proceso como tareas de mantenimiento en cada iteración, puede incluir la corrección de defectos, llevar a cabo mejoras y resolver problemas de soporte en general.

4. Caso de Estudio

En este trabajo vamos a considerar el desarrollo de software en el que participan equipos de entre 10 y 12 personas durante un período de entre 10 meses y 24 meses. La aplicación tiene una complejidad de desarrollo media-alta, esto debido tanto al origen del negocio como a la tecnología utilizada. Durante el período de desarrollo se utilizó una metodología de desarrollo ágil.

Si bien una de las características de las metodologías ágiles es la falta de énfasis en la documentación formal y mayor énfasis en la comunicación entre los miembros del equipo y el cliente, debido al tamaño del proyecto y al número de integrantes del equipo, se integró una herramienta colaborativa de gestión de conocimiento, donde se definieron actividades, artefactos, documentación y notaciones a ser utilizadas, orden de ejecución de las actividades que ayudaron desde el inicio hasta la finalización en la construcción del producto.

Como todo proceso evolutivo, a medida que se avanzaba en el desarrollo, se realizaron cambios sobre funcionalidades ya implementadas, haciendo que en algunos casos la funcionalidad se comporte de diferente manera y en algunos casos, entregando resultados diferentes. Todos los cambios, defectos encontrados y mejoras, fueron siendo documentados en la herramienta de gestión de conocimiento.

Una vez finalizada la etapa de desarrollo y con el producto finalizado y en funcionamiento, se decidió llevar a cabo un período de mantenimiento utilizando también un proceso ágil. Para ello, tuvieron que ajustarse algunas pautas o reglas al momento de realizar mantenimiento evolutivo de software, propias de las metodologías ágiles. Uno de los objetivos consistía en colaborar con el equipo de trabajo para que puedan tomar las decisiones correctas, con la mayor cantidad de información posible.

Experiencia realizada con el mantenimiento Una lista de factores que se tuvieron en cuenta en el proceso de mantenimiento del caso de estudio propuesto, sobre los que hubo que realizar algunos ajustes o adaptaciones, sin comprometer el espíritu del modelo de proceso ágil respetando las características de incremental, cooperativo, sencillo y adaptativo tal como lo definió Abrahamsson [1]. Los aspectos considerados son los que se detallan a continuación:

1. Rotación de recursos en el equipo: Durante el ciclo de vida del sistema de software, el conocimiento obtenido con la experiencia de los desarrolladores y quienes han tenido la responsabilidad de mantenerlos en funcionamiento, se pierde una vez que estos dejan la organización o son asignados a otros proyectos. Si consideramos que con frecuencia es personal sin experiencia el asignado a las nuevas tareas de Mantenimiento de Software, es un factor a considerar al momento de armar el equipo ágil de mantenimiento.
2. Falta de conocimiento del negocio, diseño y arquitectura. Falta de conocimiento explícito y peor aún tácito, por parte del equipo de mantenimiento. Esto puede deberse a la falta de experiencia de los integrantes del equipo, pobre traspaso de

información por parte de integrantes que ya no pertenecen, sumada a una escasa documentación, o distribuida en diferentes documentos o diferentes herramientas de gestión de conocimiento.
3. Escasa documentación: La mayoría de la información del producto y sus características la conoce el equipo de desarrollo, pero no está registrada en algún documento de diseño.
4. Reuniones periódicas o por demanda: Estas reuniones reemplazan a las reuniones diarias, ya que se trabaja con un equipo reducido con respecto al original y también porque parte de las tareas a realizar en la iteración actual son de corta duración.
5. Cambios en priorización en la iteración actual: (Mantenimiento Correctivo vs Evolutivo): Cuando existe un solo equipo de trabajo, si se detecta algún defecto en la aplicación que está en producción, (teniendo en cuenta la criticidad y el impacto del mismo), alguno de los integrantes del equipo debe dejar pendiente las tareas relacionadas con la iteración actual, y llevar a cabo la corrección, y luego de las pruebas correspondientes, liberar la nueva versión al ambiente de producción. Estas cuestiones pueden demorar la iteración actual o bien llegar pero no con la calidad que se pretende. Se valora en la etapa de mantenimiento, la rápida y eficiente reacción ante cambios abruptos de prioridades como consecuencia de un defecto de alto impacto y criticidad en el producto final.
6. Duración de las iteraciones: La duración, puede variar de iteración a iteración. Puede ocurrir que, relacionado con el punto anterior, haya un cambio de prioridad ante un defecto detectado en el producto en producción, y deba liberarse una nueva versión del producto (*release*) con la solución de ese defecto en cualquier punto de la iteración actual.
7. Iteraciones vs Releases: Coordinación entre los integrantes del equipo en los pasajes a producción, ocasionados por el mantenimiento correctivo en cualquier momento de la iteración; marcado esto por la criticidad del defecto (*release*), con aquellos que corresponden con las tareas planificadas para la iteración actual, como parte de la evolución del sistema. Deben tenerse en cuenta los *branches* en los que se trabaja, ser cuidadosos con las fusiones del código.
8. Antes de cada pasaje a producción del producto, debe llevarse a cabo una regresión, y verificar que no se han introducido nuevos defectos, asegurando la calidad del producto.

Si bien uno de los principios del Manifiesto Ágil [3] expresa "Desarrollar software que funciona más que conseguir una buena documentación", es necesario comprender que se debe diferenciar la construcción de modelos con el único fin de documentar el producto generado, de la elaboración de modelos como proceso de diseño de soluciones.

En el caso particular del caso estudiado, debido al tamaño y complejidad del mismo en la etapa de desarrollo, se generó documentación, soportadas en una herramienta de gestión de conocimiento colaborativa. Se propuso no documentar la aplicación completa, sino solamente cuando fueran útiles y necesarias para definir la solución adecuada. Es decir, lineamientos y actividades que faciliten la construcción del producto a partir de la creación de historias de usuario funcionales y no funcionales, wikies, tareas, defectos encontrados, sin perder las características enunciadas en el Manifiesto Ágil [3] y como manera de dar soporte posteriormente a las actividades de mantenimiento.

El desafío más grande es ser lo suficientemente maduros como organización para asumir como propias las prácticas ágiles y evolucionar la forma de trabajo adaptándose a las exigencias cambiantes del entorno sin descuidar la calidad de la entrega en cada iteración.

Un problema significativo fue la recuperación de la información y todo el *background* generado durante el desarrollo del producto. Este es un punto crítico, ya que podemos encontrarnos con diferentes obstáculos que impidan, retrasen o haga en el peor de los casos, que se agreguen involuntariamente defectos en la aplicación que ya está en producción. En especial teniendo en cuenta que no todo el equipo original que llevó a cabo el desarrollo, es el asignado en la etapa de mantenimiento.

5. IRC y Mantenimiento

La figura 3 muestra las actividades involucradas en el mantenimiento ágil de software para proyectos que tienen un período de duración superior al año, utilizado en el caso de estudio en cuestión. Se explicará aquí qué decisiones se tomaron para asegurar el menor impacto posible sobre el producto ya liberado a producción durante la etapa de mantenimiento.

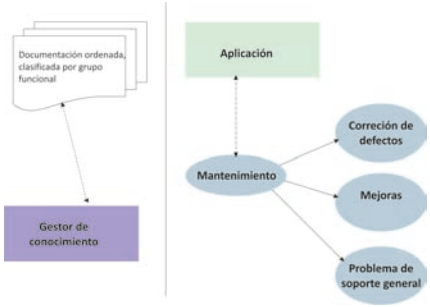

Figura 3. Mantenimiento en el desarrollo ágil

El lado derecho de la figura 3 muestra cuáles son las responsabilidades del equipo de mantenimiento que fueron divididas en tres categorías:

- Corrección Defectos: son aquellos errores que pueden ser causados por fallas en el código, por datos erróneos, o por cualquier número de factores externos.
- Mejoras: Son requerimientos solicitados por los usuarios finales, que surgen del uso del producto, y por ello aparecen nuevas necesidades, estas pueden ser modificaciones del producto ó nuevas funcionalidades, que les permitirán obtener mejores resultados en el uso diario del producto generado.
- Problemas de soporte general: Esta categoría abarca todo aquello que no es cubierto por las dos categorías anteriores. Por ejemplo análisis de performance, consultas a la base de datos, manejo de la entrega (*delivery*) de nuevas versiones al cliente, entre otros.

El lado izquierdo de la figura 3 muestra la incorporación de la herramienta utilizada en la gestión de conocimiento, en la que el equipo ágil soporta las tareas de mantenimiento. En esta herramienta se encuentran las historias de usuario, wikies, decisiones

del diseño, diseños de casos de test, documentación con los pasos para realizar los pasajes a producción, y todas las tareas que formaron parte de los pedidos pendientes (*backlog*) del desarrollo del producto.

Debido a la velocidad con la que se trabaja utilizando estas metodologías, y teniendo en cuenta que entre los valores del Manifiesto Agil [3] se promueve la comunicación verbal entre los integrantes del equipo por sobre la documentación, y que hay que dar respuesta a los cambios, sobre el cumplimiento de un plan, puede esto convertirse en una desventaja, ya que la información del producto y sus características existen en la mente de los desarrolladores, pero no quedan en un documento de diseño, con algún tipo de ordenamiento que pueda ser utilizado posteriormente por un equipo de mantenimiento.

Este tipo de situaciones, puede generar en los integrantes del equipo cierta incertidumbre, falta de seguridad y confianza al momento de realizar por ejemplo, un cambio en el diseño, efectuar una mejora, solucionar un defecto o estimar tareas que pueden poner en riesgo el éxito en esta etapa. Decisiones tomadas durante el diseño acompañado por su escasa de documentación, pueden llegar a confundir o desconcertar al equipo de mantenimiento, al no comprender el porqué de ciertas decisiones que fueron tomadas. Esto suele ocurrir con equipos de mantenimiento en los que la mayoría de sus integrantes no formaron parte del equipo de desarrollo original.

Para mitigar esta situación, se propuso a manera de experiencia, organizar, ordenar y reagrupar el conocimiento repartido entre el gestor de conocimiento, mails, documentación en general del proyecto.

La propuesta se basa en que, antes de iniciar el proceso de mantenimiento propiamente dicho, se lleve a cabo una iteración denominada Iteración de Reordenamiento del Conocimiento (IRC). La Figura 4 en el lado izquierdo muestra los incrementos realizados durante el período de desarrollo del sistema, los despliegues realizados en cada iteración, donde en cada incremento se tienen en cuenta los requerimientos iniciales para desarrollar el producto, y los nuevos requerimientos que van surgiendo a medida que se evoluciona para llegar al producto final. La Figura 4 muestra cómo se integra esta práctica no ágil (IRC), al proceso de mantenimiento ágil, como una iteración de transición entre la finalización de la etapa de desarrollo y el inicio de la etapa de mantenimiento.

La duración de esta iteración debe tomar entre una y dos semanas aproximadamente. Su objetivo es el de organizar, depurar la información que se ha registrado en el gestor de conocimiento, incorporar aquella información que no se encontraba accesible, completar información relevante de algunas wikies, marcar las historias de usuario como obsoletas, eliminar información confusa, vincular tareas que han tenido relación entre sí, agregar etiquetas a las wikis de manera de organizarlas por grupo funcional. Esto facilita futuras búsquedas, haciéndolas más efectivas en cuanto a los resultados que se obtienen.

El tiempo invertido en la reestructuración del conocimiento, fue capitalizado durante el período de mantenimiento propiamente dicho, ya que se logró que en cambios y mejoras de algunas funcionalidades, e incluso de diseño, se recuperara la información requerida, las wikies actualizadas con información confiable, como base para comprender el alcance, el impacto y el riesgo en los cambios y mejoras propuestas.

La IRC se incorpora como una práctica no ágil, implementando una adaptación híbrida en el proceso de mantenimiento ágil de software. Comprobamos que en un desarrollo ágil que llevó más de un año en su implementación, durante el período de mantenimiento se redujeron algunos efectos negativos. Decisiones de cambios en el diseño, cambios en funcionalidades, mejoras de performance, fueron tomadas sobre una

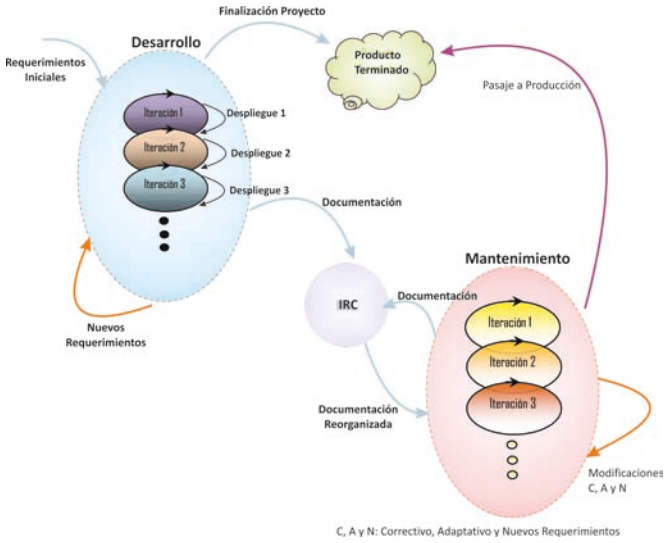

Figura 4. Mantenimiento con IRC

base más sólida de conocimiento documentado, reagrupado y depurado al implementar IRC. Se redujo el impacto y riesgo de algunos cambios durante el mantenimiento, sin perjuicio de perder la agilidad del proceso.

6. Conclusiones

La industria de desarrollo de software sufre retrasos en la finalización de proyectos, debido a los requerimientos de documentación pesados de los modelos de procesos tradicionales. Como contrapartida, los modelos de desarrollo ágil se caracterizan por el poco peso que tiene la documentación *detallada* durante el proceso de desarrollo, y la importancia de la intercomunicación entre los integrantes del equipo. Hemos descripto anteriormente cómo en el gestor de conocimiento, la información suele quedar desactualizada, insuficiente o incompleta debido a la velocidad con que se trabaja en estas metodologías y a los frecuentes cambios en los requerimientos. Esta situación termina impactando en el proceso de mantenimiento de software utilizando metodologías ágiles, sobre proyectos que tienen una duración de más de una año. El entendimiento de la traza del código, termina siendo uno de los principales puntos en la etapa de mantenimiento del software, y en ocasiones acompañado por la falta de comprensión del porqué de ciertas decisiones tomadas en su diseño.

En este trabajo, se presentó una práctica no ágil, que adaptamos al modelo ágil. Consiste en una Iteración de Reordenamiento del Conocimiento (IRC) que se lleva a cabo antes de iniciar el proceso de mantenimiento. La propuesta consiste en reorganizar el conocimiento clave, que aporte valor directo al producto, sin llegar a escribir ni reescribir requerimientos detallados. El uso de esta práctica híbrida antes de iniciar la etapa de mantenimiento, permitió que el equipo alcanzara un buen rendimiento, sobre todo en lo concerniente a cambios en las funcionalidades, cambios en el diseño, cambios en

la arquitectura, implementación de nuevas funcionalidades, donde resultó más sencillo encontrar información relacionada, que se encontraba en el gestor de conocimiento. La combinación y adaptación de ciertas prácticas utilizadas en metodologías tradicionales con metodologías ágiles, permiten obtener procesos híbridos, con los beneficios de ambas metodologías. Las organizaciones dedicadas al desarrollo de software, no siempre deben adoptar el modelo ágil en sus procesos, sino que deben adaptarlo a sus procesos y necesidades.

Si bien no se recomienda utilizar metodologías ágiles en desarrollos que excedan los 8-12 meses, la industria del software las adopta cada vez con mayor frecuencia, sin importar la duración de los proyectos. Esta práctica híbrida (IRC), podría adoptarse durante este proceso cada cierto período de tiempo (mayor a ocho meses), también por demanda del equipo. Esta decisión puede tomarse si se tienen errores en el diseño, o se introducen defectos que impactan en el comportamiento del producto, debido a la falta de conocimiento de ciertos detalles del diseño, la implementación o el negocio.

Referencias

1. P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta. *Agile Software Development Methods - Review and Analysis.* VTT Elektroniikka, 2002.
2. A. Abran and H. Nguyenkim. Measurement of the maintenance process from demand-based perspective. *Journal of Software Maintenance. Research and Practice.*, 5(2):63–90, 1993.
3. K. Beck, J. Grenning, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. *Agile Manifesto*, 2001.
4. International Organisation for Standarization. *International Organisation for Standarization. Software Engineering - Software Life Cycle Processes - Maintenance, ISO/IEC Standard 14764.*, 2006. International Organisation for Standarization: Geneva, Switzerland.
5. M. Kajko-Mattsson. Problems in agile trenches. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ESEM '08, pages 111–119. ACM, 2008.
6. D. Rico. *Agile Methods and Software Maintenance.*, 2008. http://davidfrico.com/rico08f.pdf.
7. P. M. Senge. *The Fifth Discipline: The Art & Practice of The Learning Organization.* Doubleday, 1990/2006.
8. S. Shaw. Using agile practices in a maintenance environment. *Intelliware Development Inc*, 2007.
9. P. Stachour and D. Collier-Brown. You don't know jack about software maintenance. *Communications of the ACM.*, 52(11):54–58, 2009.
10. H. Svensson and M. Host. Introducing an agile process in a software maintenance and evolution organization. In *Proceedings of the Ninth European Conference on Software Maintenance and Reengineering*, CSMR '05, pages 256–264. IEEE Computer Society, 2005.
11. H. Svensson and M. Host. Introducing an agile process in a software maintenance and evolution organization. In *Proceedings of the Ninth European Conference on Software Maintenance and Reengineering*, CSMR '05, pages 256–264. IEEE Computer Society, 2005.