Propuesta Metodológica para el Desarrollo de Software de Investigación

Methodological Proposal for Software Development Research

Héctor Antillanca Espina
Departamento de Ingeniería Informática
Universidad de Santiago de Chile
Santiago, Chile
hantilla@informatica.usach.cl

Gerardo Cerda Neumann
Facultad de Ingeniería y Tecnología
Universidad UCINF
Santiago, Chile
gcerda@ucinf.cl

RESUMEN

El software de investigación es una aplicación que tiene por objetivo probar una técnica, simular un comportamiento de interés, apoyar la generación de un *paper* o lograr una patente. En definitiva: busca generar conocimiento. Para construir estas aplicaciones se generan proyectos de desarrollo de software en un contexto científico, por ejemplo en un laboratorio de investigación de una empresa cuyo negocio es la búsqueda eficiente en la Web, un proyecto podría ser la búsqueda de la mejor técnica de recuperación de datos para operar bajo ciertas condiciones de trabajo. El equipo de desarrollo lo forman personas de alto nivel profesional, generalmente es multidisciplinario y cabe dentro del concepto de *equipos auto dirigidos*. Sin embargo estos desarrollos tienen casi las mismas dificultades que la producción de software para contextos comerciales por ejemplo dificultad para cumplir con la entrega, escasa documentación, problema para coordinar el Equipo de Trabajo y una gran complejidad para modificar el código escrito por otros desarrolladores. El objetivo buscado es proponer un método práctico para mejorar la calidad del producto construido así como la documentación generada.

Palabras clave: desarrollo de software, software de investigación, método de desarrollo, métodos ágiles, equipos autodirigidos.

ABSTRACT

The software of research is an application that aims to test a technique to simulate a behavior of interest, support the generation of a paper or get a patent. In short: aims to generate knowledge. To build these applications are generated software development projects in a scientific context, for example in a research laboratory of a company whose business is the efficient search on the Web, a project might be searching for the best data recovery technique to operate under certain conditions. The development team are highly professional people, usually multidisciplinary and fits within the concept of self-directed team. However, these developments have almost the same difficulties as the production of software for commercial settings such difficulty meeting delivery, poor documentation, a problem to coordinate the Task Force and a lot of complexity to modify the code written by other developers. The objective sought is to propose a practical method to improve the quality of the product built and generated documentation.

Keywords: software development, software of research, development methods, agile methods, self-directed teams.

INTRODUCCIÓN

El desarrollo de software de investigación posee características muy diferentes del desarrollo de software comercial. Normalmente el software de investigación se realiza en ambientes de investigación científica, donde los mismos integrantes del equipo de desarrollo están interesados en los productos y en general son los usuarios finales de las aplicaciones. Por la misma razón los requerimientos son definidos por el mismo equipo de trabajo. En general, ellos invierten muy poco esfuerzo en el desarrollo de la interfaz de usuario ya que al ser los usuarios de las aplicaciones asumen la responsabilidad de ocuparlas bien. Se observa que los miembros del equipo participan de una cultura común que los lleva a no documentar lo obvio, aunque esa práctica con el tiempo puede llegar a ser una dificultad cuando se produce rotación en el equipo. En ocasiones existe la necesidad de procesar grandes volúmenes de datos lo que obliga a constantes ajustes y mejoras a las aplicaciones desarrolladas. Lo anterior trae como consecuencia que se generen múltiples versiones de las aplicaciones en varios computadores distintos cada uno con sus datos de ejecución. La dificultad aparece al

querer saber qué resultado se generó con qué versión y cuáles fueron los datos procesados.

Ejemplos de este tipo de software son aquellos que simulan motores de búsqueda, como las que se realizan en los laboratorios de investigación de empresas tales como Yahoo! y Google, o como los simuladores de modelos científicos donde se busca encontrar algún tipo de mejora u optimización en el funcionamiento, como los que se desarrollan en laboratorios investigación de muchas universidades del mundo.

Observamos que, en general, un software de investigación tiene atributos diferentes de los que se encuentran en un software comercial. Muchos de los atributos que se espera que estén presentes en un alto grado en las aplicaciones comerciales, en el software de investigación no son importantes. Por ejemplo, no es importante que el software proteja los datos contra usuarios maliciosos, tampoco son importantes la portabilidad, la usabilidad y la tolerancia a fallos, entre otros. Sin embargo, es deseable que el software sea correcto, eficiente, mantenible. Lamentablemente, en la práctica, no todos los atributos deseables están presentes en un software de investigación, pues también dependen del proceso de desarrollo que se utilice para su producción. Por ejemplo, si el proceso no establece actividades para observar estándares de diseño y documentación, es muy probable que el producto final no sea mantenible, es decir, no será fácil de modificar por otros que quieran hacerlo en un tiempo distinto o en otro contexto.

Una primera conclusión con respecto a este tipo de proyectos es que su desarrollo está guiado por el conocimiento que se va logrando con ellos. En efecto, al iniciarlos no está totalmente definido ni lo que se obtendrá ni el aprendizaje que se logrará con el mismo, pero claramente sigue un esquema evolutivo.

Sin embargo se presentan los mismos problemas que en cualquier desarrollo: dificultad para cumplir los plazos de entrega, escasa documentación, difícil coordinación del equipo de trabajo y tiempo excesivo para modificar el código escrito por otros desarrolladores. Debido a lo anterior es que se realiza esta propuesta metodológica que permitirá aumentar las probabilidades de éxito de este tipo de proyectos.

Para realizar esta propuesta, este artículo ha sido organizado de la siguiente manera: en la siguiente sección describimos el método utilizado para conocer el problema y desarrollar la propuesta metodológica, luego presentamos los resultados del análisis del problema, posteriormente presentamos el método de desarrollo de software científico. La presentación metodológica termina con una descripción de los conceptos de equipo autodirigido pertinentes para este caso. En las secciones finales describimos brevemente los resultados, alcance y conclusiones.

MÉTODO UTILIZADO

Se entrevista a los participantes en este tipo de proyectos para analizar sus necesidades y requerimientos con el fin de proponer un método que sea práctico para ellos.

A partir de estas entrevistas es posible identificar todos los pasos necesarios para este tipo de proyectos así como las principales dificultades que se repiten constantemente. Con esta información se pudo analizar la situación actual y realizar la propuesta metodológica.

Finalmente se hace una investigación bibliográfica para fundamentar la propuesta de solución.

SITUACIÓN DE DESARROLLO ACTUAL

A. Características

Los proyectos de desarrollo de software de investigación se inician por la necesidad de evaluar una técnica de procesamiento de datos a partir del análisis de un área de conocimiento específico en estudio. Debido a esto se define un equipo de trabajo cuyo objetivo es construir un software que implemente el tema en estudio. Esto define un proyecto que tiene una duración aproximada de un mes.

Los hitos genéricos para este tipo de proyectos son:

- Definir el problema que se desea investigar o la técnica que se desea probar.
- Asignar los recursos necesarios al proyecto y definir quien lo liderará.
- Hacer un estudio inicial de las técnicas que se utilizarán. Eventualmente generar algunos prototipos para probar dichas técnicas.
- Realizar una investigación sobre los conceptos que se van a aplicar.
- Definir el experimento que se va a realizar así como los datos que se utilizarán.
- Construir el software que se necesita para realizar el experimento.
- Procesar los datos con el software construido y extraer las conclusiones.
- Escribir el paper a partir de los resultados obtenidos.
- Eventualmente repetir el experimento modificando el software o los datos utilizados. Dependiendo de los resultados se puede modificar el paper escrito.
- Presentar el paper o informe técnico generado a una instancia de revisión superior a la del equipo de proyecto.

Los resultados del proyecto normalmente se publican en alguna instancia académica relacionada con la temática estudiada.

Este tipo de proyectos tienen las siguientes características:

- El software es una herramienta para aprender, donde el paradigma habitual es simular el problema.
- El software construido es ocupado solo por usuarios especializados y no tiene el concepto de "Cliente" que define y valida los requerimientos. Los mismos integrantes del proyecto definen y utilizan posteriormente el software generado.
- El software construido está orientado al apoyo de la investigación y generación de reportes técnicos o papers.
- El software construido posee interfaz humano – computador más bien simple y cuyos usuarios son pocos y especializados.
- En general no se dedica mucho tiempo a la documentación lo que dificulta la revisión del software construido en el futuro. Generalmente cuando se realiza una modificación al código debe ser realizada por quien codificó inicialmente la aplicación ya que significa mucho tiempo explicarle a otra persona para que lo haga. Se produce entonces el fenómeno del "software de Juan" pasando estos a tener en la práctica verdaderos "dueños".
- Los proyectos al avanzar en su desarrollo dan origen a otros subproyectos donde se analizan y exploran ideas surgidas del trabajo original. Algunos de estos subproyectos son liderados por el mismo equipo original y en otros casos se asignan otros distintos.
- Pueden surgir cambios muy bruscos en los requerimientos de las aplicaciones que se han definido lo que provoca dificultad para reenfocar los recursos y los esfuerzos del equipo de trabajo.
- Cada proyecto intenta demostrar una hipótesis específica pero durante su desarrollo suelen surgir otras más que pueden ser comprobadas o no. La comprobación de cada hipótesis debe ser correctamente documentada para futuras referencias, considerando las versiones de software y de datos utilizadas para ello.

Respecto a los desarrolladores se puede decir que:

- Si bien son muy buenos programadores no tienen las costumbres que propone la Ingeniería de Software en cuanto a seguir un método específico de desarrollo. El tipo de proyecto tiene similitud con la Programación Exploratoria (*desarrollo de un sistema que se pone en marcha tan pronto como sea posible, para luego ser modificado hasta que su desempeño sea adecuado. Usualmente esta aproximación se utiliza en el desarrollo de sistemas de inteligencia artificial (AI) en que los usuarios no pueden formular una especificación de requerimientos detallada y donde la adecuación más que la correctitud es el objetivo de los diseñadores* (tomado de [12]).
- Poseen una buena disposición para incorporar mejoras en el proceso de desarrollo de software siempre que sean fáciles y rápidas de usar.

B. Dificultades

Uno de los principales problemas es la dificultad de reproducir los experimentos realizados. La dificultad aumenta en proporción al tiempo transcurrido. Generalmente no se registra qué versión de la aplicación generó los resultados publicados ni los datos utilizados.

Es difícil planificar este tipo de proyectos y definir sus fechas de término y de entrega de avances. A veces se asignan recursos insuficientes a proyectos de cierta complejidad.

En general los errores que se cometen en algunos proyectos se repiten en otros ya que se documenta escasamente lo realizado, impidiendo de esta forma crear una base de conocimiento común.

Resulta difícil modificar las aplicaciones ya construidas debido a que la documentación de las mismas es más bien escasa. En general solo se documenta en el mismo código escrito pero sin tener un estándar definido y aceptado para eso. En consecuencia depende de quien haga el código la calidad de la documentación generada. Por las mismas razones es complejo modificar una aplicación desarrollada por otro equipo de proyecto.

Por último cabe destacar que resulta difícil manejar múltiples versiones de los papers generados así como hacer el seguimiento entre las distintas versiones generadas y los programas utilizados y los datos procesados.

Para superar estas dificultades se presenta a continuación una propuesta de desarrollo.

PROPUESTA DE MÉTODO DE DESARROLLO

A. Cómo superar las dificultades

La Ingeniería de Software se ocupa de las teorías, métodos y herramientas para el desarrollo del software profesional [9]. Incluye en el software las aplicaciones desarrolladas así como la documentación generada. Se espera que los ingenieros de software usen un enfoque sistemático y organizado en su trabajo. También se espera que utilicen herramientas y técnicas apropiadas dependiendo del problema que se va a resolver, de la restricción del desarrollo y de los recursos disponibles.

Como se habla de ingeniería se da por sentado de que es necesario llegar a un equilibrio entre el esfuerzo realizado y la calidad del producto obtenido. En este sentido en los últimos años se han desarrollado dos enfoques diferentes, aunque no contradictorios.
Por un lado se ha trabajado en metodologías que definen de manera exhaustiva el uso de herramientas, documentación y notaciones, como es el caso de RUP (Rational Unified Process) [4]. Fue desarrollada por los mismos creadores de la notación que ya representa un estándar de facto en la creación de software, el UML [10]. Esta metodología ha sido ampliamente estudiada y documentada, incorporándose como estándar en muchas empresas [6]. RUP ha tenido una gran difusión ya que incorpora varias de las mejores prácticas desarrolladas en la industria: desarrollo iterativo (o en espiral), manejo de los requerimientos, uso de una arquitectura de componentes, modelamiento visual del software, verificación de la calidad y control de cambios [11].

Por otro lado se han desarrollado metodologías orientadas a la creación del código rápidamente y centradas en la evolución y adaptación, como son las llamadas metodologías ágiles [7], destacándose entre ellas XP y Scrum por lo extendido de su uso [3]. La palabra ágil quiere decir que son metodologías que se adaptan muy bien a los cambios en el entorno de desarrollo [3].

Para apreciar la diferencia entre estos enfoques la Tabla 1 muestra una comparación entre los métodos RUP y Scrum [7].

Tabla 1: Comparación entre RUP y Scrum

Característica	RUP	Scrum
Construcción de código	En forma individual	En pareja
Documentación generada	Gran cantidad de artefactos y elementos de documentación	Pocos elementos para modelar y documentar
Generación de versiones preliminares del software	Lo maneja	No lo maneja
Manejo del cambio de requerimientos	Se agregan más iteraciones para incorporar los cambios necesarios	Se espera el cambio y se maneja de forma inmediata
Participación del Cliente	Se relaciona con el equipo de desarrollo	Es parte integral del equipo de desarrollo
Participantes	Se definen variados roles según la etapa e iteración que se esté ejecutando	Se definen pocos roles que se mantienen durante el proyecto
Tamaño de los proyectos	Todo tipo de tamaños pero especialmente medianos y grandes	Generalmente proyectos pequeños

Como se puede apreciar el método RUP se presta mejor para el desarrollo de software donde haya un rol más tradicional del cliente y se necesite generar una documentación muy completa. Por su parte Scrum se presta mejor para proyectos como los que se están analizando, es decir donde no existe una clara separación entre el cliente y el equipo de desarrollo y no se desea una documentación muy extensa. De hecho se presta muy bien para el desarrollo de software hecho para pocos usuarios ya que no exige documentar el uso de manera tan exhaustiva. También se debe considerar que en las entrevistas realizadas los participantes de los proyectos se han manifestado reacios a crear documentación muy extensa respecto a las pruebas realizadas al software construido. Los métodos ágiles se prestan mejor a la programación exploratoria que es un modelo que describe bien a la mayoría de los proyectos que se realizan en el contexto descrito.

Debido a los antecedentes analizados se decidió realizar la propuesta metodológica basándose en las metodologías ágiles.

B. Propuesta Metodológica

Cabe destacar que una propuesta metodológica tiene 4 elementos [8], [7]:

Principio: también denominado "filosofía de la metodología", es la norma o idea fundamental que rige el pensamiento o la conducta, y orienta el análisis, diseño y desarrollo del software. Es el Principio, el que ordena y estructura las herramientas que son aplicables en la metodología, así como los Procedimientos con los que se aplica. Tradicionalmente, se apellida a cada metodología en función del principio que la rige.

Herramientas: son definiciones de mecanismos manuales, semiautomáticos o automáticos que permiten analizar, diseñar o construir el software. Las herramientas quedan estrechamente ligadas al principio rector de la metodología y es muy poco probable que una misma herramienta sea utilizable en más de una metodología. Una herramienta debe tener un objetivo específico y un método de aplicación. Por lo general, se ha demostrado que las herramientas gráficas (que usan imágenes) son más fáciles de usar y entender que las herramientas que sólo se sustentan en textos escritos. Son ejemplos de herramientas: los DFD Diagramas de Flujo de Datos, MER Modelos Entidad Relación, Lenguaje Estructurado, Diagramas de Componentes, Diagramas de Herencia, etc.

Modelos: el modelo define las etapas a realizar para alcanzar la solución al problema planteado. Los Modelos, se refieren a la forma de organizar los Procedimientos, de manera de obtener resultados de calidad en el menor tiempo posible. A diferencia de las Herramientas y los Procedimientos, los modelos son relativamente independientes del principio, pudiendo

aplicarse sin grandes dificultades, cualquier modelo a cualquier metodología. Pese a lo anterior, el modelo debe quedar definido claramente antes de iniciar el desarrollo del software. Ejemplos de modelos son: Cascada, Prototipos y Espiral.

Procedimientos: se refiere al modo de hacer, con orden, las cosas; es decir, como poner en práctica las herramientas. Los procedimientos corresponden a la definición que permite unir y ordenar los resultados de cada herramienta y facilitan el desarrollo racional y oportuno de software. Definen la secuencia en la que se aplican las herramientas, la entrega de los resultados de ellas, los controles que ayudan a asegurar la calidad. También coordinan y controlan los cambios y entregan las directrices que ayudan a los administradores a evaluar el progreso del proyecto.

Finalmente cabe destacar que la metodología, para que sirva a nuestros propósitos, debe cumplir con dos condiciones fundamentales:

- Debe tener Hitos bien definidos: el analista/diseñador debe saber claramente cuales son los objetivos de la etapa en la que se encuentra, reconociendo claramente las tareas que debe realizar para alcanzar dichos objetivos.
- Debe ser Incremental: el resultado de una etapa, debe ser de utilidad para la persona que va a realizar la etapa siguiente.

La propuesta metodológica lleva por nombre PDKD o Project Driven Knowledge Discovery, es decir Proyecto Guiado por el Descubrimiento del Conocimiento.

La propuesta tiene los siguientes elementos en consecuencia:

Principio: corresponde al hecho de que el proyecto de desarrollo está guiado por el descubrimiento de conocimiento. Por esta razón al finalizar cada vuelta del ciclo se analiza si se continúa con el proyecto o se genera uno nuevo. Los ciclos continúan hasta que se ha logrado el conocimiento buscado. El principio es poder repetir los experimentos cada vez que sea necesario independiente del tiempo transcurrido y de los participantes en el proyecto.

Herramientas: se definen las siguientes herramientas de apoyo:

- Reuniones diarias de coordinación de 15 minutos "de pie". Cada integrante responde las preguntas: ¿qué hice ayer?, ¿qué voy a hacer hoy? y ¿qué dificultades he tenido?
- Una Wiki del proyecto para los compromisos establecidos.
- Para la documentación se sugieren las siguientes aplicaciones:
  - El generador automático Doxygen para el código, actualmente en su versión 1.5.2.
  - El sistema de control de versiones SVN (Subversión) para el software y la documentación.
- Una "Matriz de Trazabilidad" donde quede registrado cada versión de software generado, los datos que procesó, los resultados que obtuvo, en que fecha, en que computador se encuentra cada versión y que Proyecto fue el responsable de la creación de la aplicación.

La Matriz de Trazabilidad documenta los siguientes detalles:

Proyecto: Proyecto 002 – 2009		**Equipo de Trabajo:** Equipo 1	
Ciclo	**Aplicación creada**	**Desarrollador:**	**Datos ejecución:**
1	Pr005	ABC, HIJ …	Pr005.dat
…			
N			

Inicio: 20/12/2009	
Fecha ejecución:	**Documento de resultados:**
01/01/2010	Inf005.doc

…

El nombre del proyecto.
El nombre del equipo que realizó el proyecto.
La fecha de inicio del mismo.

Para cada ciclo se debe registrar:
- La aplicación creada identificada con un nombre único.
- El o los desarrolladores responsables.
- Los datos de ejecución utilizados, identificados de forma única.
- La fecha de ejecución de la aplicación con esos datos.
- Los documentos de registro y análisis resultantes.

Cada Equipo de Trabajo define la ubicación de sus aplicaciones, datos de ejecución y documentos de resultados. Estas definiciones son conocidas por todos los equipos.

Modelo: espiral, tal como se aprecia en la Figura 1.

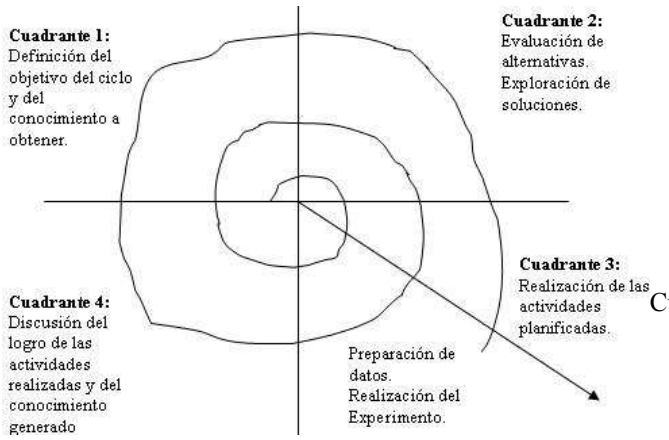

Figura 1: Modelo propuesto en espiral (adaptado de Sommerville [9])

Cada vuelta, de aproximadamente dos semanas va a permitir analizar el cumplimiento de los requisitos y la posible creación de otro proyecto, derivado del primero.

Para cada vuelta de la espiral se puede utilizar la siguiente plantilla adaptada de [9]:

- **Objetivos:** describe la intención del ciclo a realizar. La guía general es incrementar el conocimiento
- **Restricciones:** factores que limitan las posibilidades de realizar el ciclo y de generar conocimiento
- **Alternativas:** diferentes maneras de lograr los objetivos
- **Conocimiento:** posibles conocimientos que se pueden obtener con el ciclo
- **Descubrimiento del conocimiento:** estrategias para generar el conocimiento
- **Resultados:** el conocimiento generado
- **Planes:** como atacar el siguiente ciclo de ser este necesario
- **Compromisos**: decisiones de cómo continuar

De esta manera será posible aumentar la calidad de la interacción dentro del equipo y el apoyo a otros proyectos en desarrollo. Este hito es llamado "Control de Versión" y permitirá evaluar el avance logrado, concluir si el proyecto continúa adelante, se deja en pausa, se cancela o da origen a otros sub proyectos.

Procedimientos: las actividades a desarrollar son las siguientes:

Cuadrante 1:

1. Definición del objetivo para el ciclo. Esta actividad está liderada por el Jefe de Proyecto pero cuenta con la participación y aporte del Equipo de Trabajo completo.
2. Asignación de tareas para los integrantes del Equipo de Trabajo.

Resultados:

- Un breve informe con el objetivo a lograr en el ciclo.
- Cada integrante tiene su actividad definida (uso de la Wiki).

Cuadrante 2:

3. Investigación previa: cada integrante realiza una breve investigación (no más de 2 días) para cuantificar los tiempos que requerirá para completar su tarea.
4. Una vez finalizadas las primeras actividades se definen los requerimientos que deberá cumplir la aplicación que se desarrollará. El Equipo de Trabajo define de común acuerdo el plazo de construcción de la aplicación y los roles específicos que cumplirá cada uno: Desarrollador-Documentador, Encargado de Datos y Tester (uso de la Wiki). Puede haber más de un integrante en cada rol.

Resultados:

- Una lista de funcionalidades y características que la aplicación deberá cumplir.
- Una asignación de los roles para cada integrante.

Cuadrantes 2 y 3:

5. Reunión de análisis: el Equipo de Trabajo analiza los pasos a seguir acordando las acciones para la siguiente semana. Las acciones se definen indicando: responsable, plazo para completarla y responsable (uso de la Wiki).
6. Se crea el código de la aplicación (uso de Doxygen y SVN). Cada integrante reporta diariamente sus avances, dificultades y descubrimiento (reunión de coordinación de 15 minutos "de pie").

Resultados:

- Una lista actividades asignadas.
- Una lista de descubrimientos realizados por los integrantes del equipo.

Cuadrante 3:

7. Creación del diseño, definición de las pruebas a realizar, codificación y realización del experimento con los datos seleccionados ("Matriz de Trazabilidad"). Se analizan los resultados.

8. Se repite el paso 7 hasta que el grupo esté satisfecho con los resultados obtenidos (uso de Doxygen y SVN).

Resultado:
- Un informe de análisis con los resultados obtenidos.

Cuadrante 4:

9. Al cerrar cada ciclo se podrá tomar la decisión de realizar otro conjunto de actividades siendo estas de los siguientes tipos:
   - Incremento de funcionalidad al software desarrollado.
   - Preparación de los datos de prueba.
10. Extraer las conclusiones del experimento realizado.
11. Evaluar la generación del paper o informe técnico para que sea revisado por gente externa al grupo de trabajo.

    A partir de esta revisión pueden ocurrir las siguientes situaciones:
    - Que sea aceptado para su publicación.
    - Que se sugieran nuevas pruebas y en consecuencia se realicen nuevos ciclos.
    - Que surjan nuevas ideas de proyecto y que se definan nuevos grupos de trabajo.
12. En caso de realizar un nuevo ciclo desarrollar los planes para la siguiente fase del proyecto.

Resultado:
- La decisión de realizar o no un nuevo ciclo.
- Las conclusiones del experimento realizado.
- Un paper o informe técnico que presenta el conocimiento generado.

En cada Equipo de Trabajo existen los siguientes roles y responsabilidades:

- Jefe de Proyecto: orientar y liderar el trabajo del equipo.
- Desarrollador-Documentador: escribir el código, documentar la programación.
- Encargado de Datos y Tester: preparar los datos a procesar, ejecutar los programas creados, revisar los resultados.

A continuación se presentan algunos conceptos que se deben tener en cuenta al utilizar equipos autodirigidos como los que se propone utilizar.

TRABAJO EN EQUIPOS AUTODIRIGIDOS

Para finalizar esta presentación metodológica se desarrollarán los conceptos principales de los equipos autodirigidos. En primer lugar se destacarán sus características principales y luego se presentará el modelo de trabajo en equipo de Dickinson y MacIntire ya que define muy bien la forma de trabajo de este tipo de equipos.

A. Características de estos equipos

Obviamente que un equipo de trabajo o de proyecto no existe solo porque alguien lo haya definido así. Para que realmente sea un equipo de trabajo debe haber cohesión, entendimiento y objetivos claros a lograr. En los equipos de desarrollo de software se cumple la misma situación, de ahí la importancia de que realmente estén bien coordinados internamente.

En este sentido un equipo autodirigido es "*un número pequeño de personas, que comparten conocimientos, habilidades y experiencias complementarias y que, comprometidos con un propósito común, se establecen metas realistas, retadoras y una manera eficiente de alcanzarlas también compartida, asegurando resultados oportunos, previsibles y de calidad, por los cuales los miembros se hacen mutuamente responsables*" [1] sic. En el contexto de este trabajo se entenderá por número pequeño de personas a un máximo de 5, cantidad no excedida en los proyectos estudiados. De igual manera el resto de las características de un equipo autodirigido se cumplen ya que los integrantes de los proyectos son personas que poseen los conocimientos necesarios y se complementan para el logro de los objetivos, obteniendo la mayoría de las veces un resultado de calidad.

Para que un equipo autodirigido sea exitoso es imprescindible que sus integrantes posean las siguientes características [1]:

- Estén dispuestos a aceptar la responsabilidad por las acciones que realizan y por los resultados que producen
- Se involucren en tareas para fortalecer al equipo, sobre todo en tareas que no son de su responsabilidad.
- Sean gente líderes de sí misma
- Sepan pedir ayuda sin complejos cuando la necesitan
- Tengan mucha seguridad personal

Como se puede concluir las personas que integran los equipos de trabajo de los proyectos analizados cumplen todas estas características, razón por la cual pueden aplicar el modelo de trabajo de Dickinson y McIntire que se describe a continuación.

B. Modelo de trabajo en equipo de Dickinson y McIntire

Este modelo de trabajo ha sido probado en proyectos que utilizan Scrum y ha obtenido buenos resultados [2]. Gráficamente se puede ver en la Figura 2:

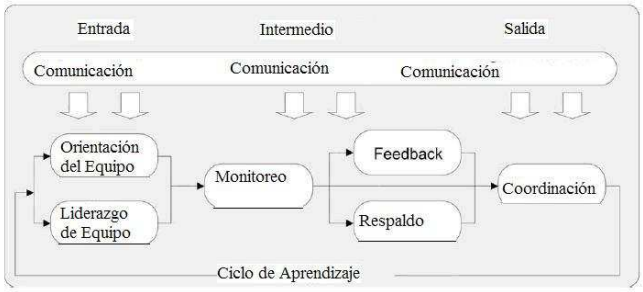

Figura 2: Modelo de trabajo en equipo de Dickinson y McIntire.

Sus elementos más importantes son:

Equipo de orientación: se refiere a las tareas del equipo y las actitudes que los miembros del equipo tienen para con los demás. Además, refleja la aceptación de las normas del equipo, el nivel de grupo cohesión, y la importancia de los miembros del equipo.

Liderazgo de equipo: proporciona la dirección, estructura, y el apoyo a otros miembros del equipo. No se refiere necesariamente a una sola persona con autoridad formal sobre los demás. Liderazgo de equipos se puede presentar en varios miembros del equipo.

Monitoreo: Se refiere a la observación de las actividades y el rendimiento de otros miembros del equipo y reconocer cuando un miembro del equipo lleva a cabo correctamente sus responsabilidades. Esto implica que el equipo tiene miembros individualmente competentes.

Feedback (Retroalimentación): consiste en la entrega, la búsqueda, y la recepción de información entre los miembros del equipo. Dar retroalimentación se refiere a proporcionar información con respecto al rendimiento de otros miembros. Buscar la retroalimentación se refiere a la solicitud de entrada o de orientación con respecto al rendimiento y la aceptación de informaciones positivas y negativas respecto al trabajo realizado.

Respaldo: implica estar siempre disponible para ayudar a otros miembros del equipo. Esto implica que los miembros tienen una comprensión de las tareas de otros miembros. También implica que el equipo de miembros está dispuesto y es capaz de ofrecer y buscar ayuda cuando sea necesario.

Coordinación: se refiere a los miembros del equipo de ejecución de sus actividades de manera oportuna e integrada. Esto implica que la actuación de algunos miembros del equipo influencia el desempeño de los demás. Esto puede implicar un intercambio de información que, posteriormente, influye en el desempeño de otro miembro.
Coordinación representa la salida del modelo y refleja la ejecución de las actividades del equipo de tal manera que los miembros deben responder en función del comportamiento de los demás.

Comunicación: implica el intercambio de información entre dos o más miembros del equipo en la forma prescrita y utilizando la terminología adecuada. A menudo, el propósito de la comunicación es aclarar o acusar recibo de la información.

El equipo de trabajo debe ser capaz de apoyarse de manera permanente dejando constancia documentada de los acuerdos tomados ya sea mediante minutas de reunión o grabaciones en video. Las interacciones posibles serán hechas de manera presencial y también en forma virtual a través de videoconferencias e intercambio de correos electrónicos.

RESULTADOS

Se dispone de una propuesta metodológica constituida por los cuatro elementos comunes a la mayoría de las metodologías de desarrollo de software: principio, herramientas, modelo y procedimientos. La propuesta considera principios de agilidad bajo un enfoque de desarrollo en espiral orientado por el descubrimiento de conocimiento. Esto permitirá enfrentar de mejor forma este tipo de proyectos ya que presenta una manera de coordinar el trabajo mediante reuniones diarias y mostrar avances periódicos usando iteraciones controladas. Por último, asumir que se está en presencia de equipos autodirigidos permite hacer explícita una forma de trabajo que busca potenciar las interacciones y los resultados de los integrantes.

ALCANCE

Las propuestas realizadas solo son efectivas para proyectos que desarrollan software de investigación, es decir cuando exista una técnica o teoría que se debe probar. Lo anterior requiere la interacción permanente y motivada de todos los integrantes del equipo quienes deben compartir conocimientos avanzados necesarios para este tipo de proyectos.

CONCLUSIONES Y RECOMENDACIONES

Se han definido y caracterizado los proyectos de desarrollo de software de investigación. El análisis de estas características ha permitido identificar las principales dificultades que existen para que logren éxito.

Se ha presentado y justificado un marco de trabajo dentro de las metodologías ágiles como propuesta de solución. Se espera que lo propuesto permita aumentar significativamente las probabilidades de éxito y aumentar la productividad de los equipos involucrados.

A continuación corresponderá aplicar esta propuesta en un proyecto real para analizar la utilidad de la misma.

AGRADECIMIENTOS

Agradecemos el aporte de las siguientes personas:

Profesor Mauricio Marín Caihuan, investigador Senior de Yahoo! Research, Santiago de Chile.

Profesor Esteban Feuerstein, líder de proyectos de investigación en Yahoo! Research.

Investigador Carlos Gómez Pantoja de Yahoo! Research, Santiago de Chile.

REFERENCIAS

[1] Ángel C., Alfredo, Trabajo en equipos autodirigidos, agosto de 2005, publicado en De Gerencia .COM, http://www.degerencia.com/articulos.php?artid=782, consultado en Julio 2010.

[2] Brede Moe, Nils, Dingsøyr Torgeir, Dybå Tore, A teamwork model for understanding an agile team: A case study of a Scrum project, The journal of Systems and Software, November 2009.

[3] Chow Tsun, Cao Dac-Buu, A survey study of critical success factors in agile software projects, The journal of Systems and Software, August 2007.

[4] Jacobson, Booch y Rumbaugh, El proceso unificado de desarrollo de software, editorial Addison Wesley, 2000.

[5] Palacio, Juan, El modelo Scrum, publicado en http://www.navegapolis.net, 2006, consultado en junio de 2010.

[6] Pressman, Roger, Ingeniería de Software, un enfoque práctico, editorial Mc Graw Hill, 2005.

[7] Prilo, Sebastián Miguel, Métodos ágiles, una alternativa real y competitiva a los procesos tradicionales de desarrollo, Manuales Users, 2009.

[8] Santibáñez, José Miguel, Fundamentos de las Metodologías en la Ingeniería del Software, artículo publicado en la revista Akádemeia, Universidad UCINF.

[9] Sommerville, Ian, Ingeniería del Software, 7ma edición, editorial Pearson – Addison Wesley, 2005.

[10] Stevens, Perdita y Pooley, Rob, Utilización de UML en Ingeniería de Software con objetos y componentes, editorial Pearson – Addison Wesley, 2002.

[11] Weitzenfeld, Alfredo, Ingeniería de software orientada a objetos con UML, Java e Internet, México, editorial Thomson, 2005.

[12] Barra Peñaloza, Carlos, Proceso y proyecto de ingeniería de software, Revista Marina, Vol. 115/845, Jul/Ago, 4, 1998..