DEL MANIFIESTO ÁGIL SUS VALORES Y PRINCIPIOS

RESUMEN
El manifiesto ágil es un documento que resume en cuatro valores y doce principios las mejores prácticas para el desarrollo de software, basados en la experiencia de 17 industriales del software, en procura de desarrollos más rápidos conservando su calidad. Este artículo presenta de manera general la evolución de las metodologías para el desarrollo de software y una reseña de los valores y principios del manifiesto ágil.

PALABRAS CLAVES: Metodologías ágiles, manifiesto ágil, desarrollo de software, metodologías pesadas, desarrollo ágil.

ELIÉCER HERRERA URIBE
Ingeniero de Sistemas.
Docente auxiliar,
Universidad Tecnológica de Pereira
eherrera@utp.edu.co

LUZ ESTELA VALENCIA AYALA
Ingeniero Industrial.
Docente auxiliar,
Universidad Tecnológica de Pereira
levayala@utp.edu.co

ABSTRACT
The agile manifesto is a document that summarizes, in four values and twelve principles, the best practices of software development based on 17 software professionals' experience, on how to develop software quickly, yet keeping its quality. This article presents, in a general manner, the evolution of methodologies used to develop software and a short view of the values and principles of said document

KEYWORDS: Agile methodologies, agile manifesto, agile development, software development,

1. INTRODUCCIÓN

En la actualidad, las metodologías ágiles se convierten en un modelo para los iniciados en el desarrollo de software, estas metodologías presentan algunas ventajas ante las metodologías pesadas pero son limitadas por el tamaño del proyecto y el número de programadores que pueden intervenir. Sin embargo, resultan muy atractivas para el desarrollo de aplicaciones en empresas de software que estén iniciando o para el desarrollo de software por módulos, sin descuidar la calidad y garantizando la actualización de la documentación.

En este artículo se presenta el concepto de metodologías ágiles. Se inicia con una breve reseña histórica del surgimiento de la computación. A continuación se expone la idea de ciclo de vida del software. Luego analizamos las metodologías pesadas que son las antecesoras de las metodologías ágiles. Finalmente, se explican los valores, principios y las implementaciones de las metodologías ágiles.

2. RESEÑA HISTÓRICA

En el siglo XIX Charles Babbage concibió y diseño la "máquina analítica"[1], la primera máquina de propósito general, la cual podía ser programada e implementaba el concepto de almacenamiento de datos, bucles e instrucciones de decisión, de modo que el programa se ejecutaba sin la intervención de una persona que la operase.

Aunque esta máquina puede considerarse, conceptualmente, como el primer computador, en realidad nunca se construyó. Por tal razón, muchos historiadores atribuyen el calificativo de "primer computador" al Z3, construido en Alemania por Konrad Zuse en los 40`s, la cual era digital, multiusos y, a diferencia de los dos primeros modelos (Z1 y Z2), plenamente funcional. [2]

De manera simultánea en Inglaterra y Estados Unidos se adelantaron proyectos de investigación que años más tarde dieron como resultado máquinas más sofisticadas y más veloces, programables, de propósitos generales y capaces de ejecutar un considerable número de tareas de mayor complejidad.

Dado que el objetivo de este artículo no es el debate dialéctico respecto a la razón que puedan tener los historiadores en cuanto a cuándo y cuál fue el primer computador y cuál fue el primer programa de computador desarrollado; se acepta el reconocimiento dado por la literatura existente sobre el inicio de la informática y la computación, en los años 50.

En la siguiente década, la programación paso de ser externa (conexión de cables y dispositivos) a ser almacenada en la memoria del computador; estos primeros programas se caracterizaron por un alto grado de dependencia entre los lenguajes, el computador y las personas que los escribían. Para enfrentar tal situación surgieron los lenguajes de alto nivel, los compiladores y los programas almacenados en medios magnéticos, haciendo el proceso de elaboración de programas más exigente, dadas las numerosas y nuevas posibilidades respecto al uso del computador.

Sin embargo, en una disciplina naciente la falta de técnicas o metodologías, hicieron evidente el exceso en los tiempos empleados, los sobrecostos, la insatisfacción de los usuarios o clientes; en definitiva, la baja calidad de

Fecha de Recepción: 24 Agosto de 2006
Fecha de Aceptación: 16 Marzo de 007

los programas. Esto fue lo que finalmente desembocó en la llamada crisis del software.

Como respuesta a esta crisis, en la segunda mitad de la década de los 60 se popularizó el concepto de ingeniería de software. Al formalizarse una metodología para la elaboración de programas bajo conceptos de ingeniería, esta tarea se transformó en un proceso riguroso propendiendo por el cumplimiento de los cronogramas, los presupuestos, y lo más importante, la satisfacción del cliente.

Durante el proceso de evolución de la ingeniería del software, se la ha concebido y definido de diversas formas, al respecto la IEEE (IEEE93), dice: "La Ingeniería de Software consiste en la aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software".

En general la ingeniería de software establece los métodos, los principios y las reglas para obtener software confiable de excelente calidad, a partir de un enfoque sistémico.

3. CICLO DE VIDA

Tradicionalmente se reconoce que, si bien, producido por el hombre, el software padece del rigor del llamado ciclo de vida, que rige a todo en la naturaleza, tiene un comienzo, un desarrollo, un proceso de maduración y un final. En el entorno del desarrollo de software, se identifica el ciclo de vida como la secuencia de: análisis y especificación de requerimientos, diseño de interfaces y de software, implementación y pruebas unitarias, de integración y del sistema, implantación, y finalmente el mantenimiento (Figura 1)

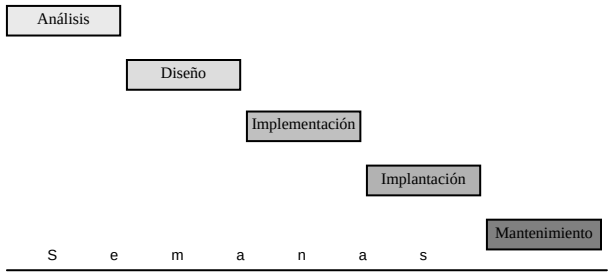

Figura1. Modelo de cascada

En su comienzo las metodologías para el desarrollo de software fueron muy claras en llamar etapas a estos conjuntos de actividades que se requieren, estableciendo un grado fuerte de discreción y secuencialidad. Cada etapa tenía un puesto en la secuencia, sus objetivos, sus entradas y salidas, sus herramientas y métodos; se podía saber en qué etapa se encontraba un proyecto.

4. METODOLOGÍAS PESADAS

Existen varios modelos para el desarrollo de software, entre los que se encuentran el modelo de cascada, el modelo incremental y el modelo en espiral. Presentamos el modelo en cascada por ser el más difundido, el de mayor popularidad, y por considerar que refleja fielmente los principios de las metodologías convencionales.

4.1 Características del modelo en cascada

Entre las más notables características del este modelo se pueden citar:

- Prestan especial atención al modelado y al mantenimiento de los modelos.
- Hacen énfasis en el control del proceso de desarrollo, en la metodología, en el rigor de las actividades involucradas en el desarrollo, en las herramientas y en la notación que se usa.
- Hacen énfasis en la especificación minuciosa del proceso, con un alto número y especialización de roles.
- Limitan la participación del cliente sólo a reuniones de control, reduciendo de manera significativa sus aportes.
- Asumen que no se van a presentar cambios una vez iniciado el proyecto y esperan que la arquitectura se defina tempranamente.
- Procuran documentar de manera rigurosa toda actividad desarrollada en el proyecto.
- Ocasionan largos tiempos de espera por parte del usuario para ver los resultados.
- Se rige por la rigurosidad de un contrato.

5. METODOLOGÍAS ÁGILES

Las metodologías ágiles resuelven los problemas surgidos, posteriormente, a la masificación del uso del computador personal, dado que las expectativas y necesidades por parte de los usuarios se hicieron más urgentes y frecuentes. Fue así como a comienzo de los 90 surgieron propuestas metodológicas para lograr resultados más rápidos en el desarrollo de software sin disminuir su calidad.

Después de casi una década de esfuerzos aislados, en febrero de 2001 en Utah-EEUU, se reunieron 17 empresarios de la industria del software[1] y como resultado del debate respecto a las metodologías, principios y valores que deben regir el desarrollo de software de buena calidad, en tiempos cortos y flexible a

[1] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland y Dave Thomas.

los cambios, se aceptó el término ágil para hacer referencia a nuevos enfoques metodológicos en el desarrollo de software.

En esta reunión se creó "*The Agile Alliance*", organización sin ánimo de lucro dedicada a promover los conceptos relacionados con el desarrollo ágil de software y acompañar a las organizaciones para que adopten dichos conceptos. Como punto de partida o base fundamental de las metodologías ágiles se redactó y proclamó *el manifiesto ágil*.

5.1. Manifiesto ágil

5.1.1 Valores

El manifiesto hace énfasis en cuatro valores principales que deben soportar el desarrollo de software:

a. Los individuos e interacciones por encima de los procesos y las herramientas: para garantizar una mayor productividad, las metodologías ágiles valoran el recurso humano como el principal factor de éxito. Reconocen que contar con recurso humano calificado con capacidades técnicas adecuadas, facilidades para adaptarse al entorno, trabajar en equipo e interactuar convenientemente con el usuario, da mayor garantía de éxito que contar con herramientas y procesos rigurosos.

Las metodologías ágiles reconocen que es más importante construir un buen equipo de trabajo que las herramientas y procesos. Procura primero conformar el equipo y que éste defina el entorno mas conveniente de acuerdo con las necesidades y las circunstancias.

b. Software funcionando por encima de la documentación: los profesionales relacionados con el desarrollo de software, aunque no es su fuerte producir documentos, reconocen su importancia, al igual que reconocen el tiempo y costo de mantener una documentación completa y actualizada.

En este sentido, las metodologías ágiles respetan la importancia de la documentación como parte del proceso y del resultado de un proyecto de desarrollo de software, sin embargo, con la misma claridad hacen énfasis en que se deben producir los documentos estrictamente necesarios; los documentos deben ser cortos y limitarse a lo fundamental, dando prioridad al contenido sobre la forma de presentación.

La documentación, en las metodologías ágiles procura mecanismos más dinámicos y menos costosos como son la comunicación personal, el trabajo en equipo, la autodocumentación y los estándares.

c. La colaboración del cliente por encima de la negociación del contrato: clásicamente el usuario o cliente es quien solicita e indica qué debe hacer el software, y espera los resultados de acuerdo con sus exigencias o expectativas, en los plazos establecidos. Con frecuencia las dos partes, cliente y equipo de desarrollo, asumen posiciones distantes, con ingredientes de rivalidad y prevención al punto de tener que dedicar tiempo valioso a la tarea de redactar, depurar y firmar el contrato.

En este sentido, y complementando el valor que se da al trabajo en equipo, las metodologías ágiles incluyen de manera directa y comprometida al cliente o usuario en el equipo de trabajo. Es un ingrediente más en el camino al éxito en un proyecto de desarrollo de software. Más que un ambiente de enfrentamiento en el cual las partes buscan su beneficio propio, evadiendo responsabilidades y procurando minimizar sus riesgos, bajo la filosofía de las metodologías ágiles se busca el beneficio común, el del equipo de desarrollo y el del cliente. La participación del cliente debe ser constante, desde el comienzo hasta la culminación del proyecto, y su interacción con el equipo de desarrollo, de excelente calidad.

Es el cliente quien sabe qué es lo que necesita o desea, el más indicado para corregir o hacer recomendaciones en cualquier momento del proyecto.

d. La respuesta al cambio por encima del seguimiento de un plan: dada la naturaleza cambiante de la tecnología y la dinámica de la sociedad moderna, un proyecto de desarrollo de software se enfrenta con frecuencia a cambios durante su ejecución.

Van desde ajustes sencillos en la personalización del software hasta cambios en las leyes, pasando por la aparición de nuevos productos en el mercado, comportamiento de la competencia, nuevas tendencias tecnológicas, etc. En este sentido, las metodologías pesadas con frecuencia caen en la idea de tener todo completo y correctamente definido desde el comienzo. No se cuenta entre sus fortalezas la habilidad para responder a los cambios.

Por el contrario, en las metodologías ágiles la planificación no debe ser estricta, puesto que hay muchas variables en juego, debe ser flexible para poder adaptarse a los cambios que puedan surgir. Una buena estrategia es hacer planificaciones detalladas para unas pocas semanas y planificaciones mucho más abiertas para los siguientes meses.

5.1.2 Principios del manifiesto ágil

Bajo el concepto de principio se hace referencia a las características que hacen la diferencian entre un proceso ágil y uno tradicional, y constituyen las ideas centrales del desarrollo ágil.

I. Nuestra mayor prioridad es satisfacer al cliente mediante entregas tempranas y continuas de software con valor. Para que una metodología puede ser calificada como ágil debe empezar a entregar software funcionando y útil en pocas semanas. Esto acaba con la incertidumbre,

desconfianza, insatisfacción y desmotivación producidas en el cliente debido a las largas esperas para ver resultados concretos.

Por lo tanto, la participación del cliente se hace más productiva en la medida en que el software esta siendo probado, revisado y aprobado constantemente por quien lo requirió y lo va a usar.

II. Bienvenidos los cambios a los requerimientos, incluso los tardíos. Los procesos ágiles aprovechan los cambios para la ventaja competitiva del cliente. Es ambicioso esperar que el cliente defina de manera definitiva todos sus requerimientos desde el comienzo y peor aún depender de ello para adelantar el proyecto.

Los cambios en los requerimientos deben asumirse como parte del proceso de maduración del software, debe entenderse que cuando el cliente describe una necesidad lo hace desde su perspectiva de usuario y que sus conocimientos técnicos lo pueden limitar para hacerse entender completamente. Por lo tanto, las novedades en los requerimientos pueden ser ajenas a la voluntad del cliente.

Esta forma de ver los cambios en los requerimientos induce al equipo de desarrollo a preferir los diseños flexibles, lo cual aumenta la satisfacción del cliente y redunda finalmente en beneficio del equipo de desarrollo dada la comodidad en el diagnóstico y ajustes que se requieren en la etapa de mantenimiento.

III. Liberar frecuentemente software funcionando, desde un par de semanas a un par de meses, con preferencia por los periodos más cortos. El cliente siempre espera ver funcionando el programa, y es eso lo que debe entregársele. Pocas veces resulta conveniente, después de varios meses de trabajo, entregar sólo informes, modelos abstractos y planes. Se deben entregar resultados que incluyan software que el usuario pueda ver trabajando. Si hay una circunstancia que motiva al cliente es poder usar el software que solicitó.

IV. Las personas del negocio y los desarrolladores deben trabajar juntos diariamente a lo largo del proyecto. Si bien el usuario desconoce lo referente al lenguaje, el diseño de bases de datos, protocolos y demás aspectos técnicos, es él, quien nos puede señalar qué está bien desde el punto de vista de la funcionalidad y resultados entregados por el software.

La intervención oportuna del usuario puede resultar decisiva en el éxito de un proyecto y puede reducir el costo o el tiempo. Esta intervención puede ser en cualquier momento, por lo cual el usuario debe estar involucrado todo el tiempo que dure el proyecto.

V. Construir proyectos en torno a individuos motivados. Darles el entorno y apoyo que necesiten, y confiar en ellos para que consigan hacer su trabajo. El ánimo, el sentido de pertenencia y la disposición del equipo de trabajo son fundamentales en un proyecto de software.

Parte de la motivación está en la confianza que se muestre en el equipo de trabajo, el respeto por sus aportes y la comodidad que se les conceda en el momento de realizar su trabajo. Todo lo que se pueda hacer por dar ánimo y motivación a las personas participantes en el proyecto debe hacerse.

VI. El método más efectivo y eficiente de compartir información a, y dentro de un equipo de desarrollo, es la conversación cara a cara. El trabajo en equipo debe apoyarse con un buen sistema de comunicación tanto entre los miembros del equipo de desarrollo como entre éstos y el usuario. La mejor forma de hacerlo es hablando personalmente; en la medida en que se evitan los intermediarios en el proceso de comunicación, como son el papel, el teléfono, el sistema de correo, y demás medios de comunicación, se incrementa la posibilidad de que el resultado sea el que se solicitó.

VII. El software funcionando es la medida de progreso. Cuando se trata de establecer el estado de un proyecto, si bien existen diversas formas de medirlo, es la cantidad de requerimientos implementados y funcionando la que mas claridad y confiabilidad ofrecen para establecer una medida del avance del proyecto. Cualquiera otra que se presente será superada por una que involucre el software qué ya ha sido probado y aprobado por el usuario.

VIII. Los procesos ágiles promueven el desarrollo sostenible. Los patrocinadores, desarrolladores y usuarios deberían ser capaces de mantener relaciones cordiales. Se debe trabajar de forma que lo urgente no se imponga sobre lo importante. Desde el inicio del proyecto se debe asignar responsabilidades y tareas de manera que siempre se puedan cumplir.

IX. La atención continua a la excelencia técnica y al buen diseño incrementan la agilidad. Además de satisfacer los requerimientos del usuario, los aspectos técnicos deben ser excelentes, independientemente de su cantidad y complejidad. La calidad debe ser vista desde dos perspectivas, la del usuario y la del equipo desarrollador. Para el personal técnico resulta evidente que cuanta más calidad tenga el software en cuanto a diseño y estándares de implementación, más rendimiento obtiene en las tareas de pruebas, mantenimiento, y mayor reusabilidad.

X. La simplicidad –el arte de maximizar la cantidad de trabajo no hecho- es esencial. Se estima que el cliente nunca usará el 90% de la funcionalidad que se implementa sin que está haya sido solicitada. Se deben centrar los esfuerzos en lo que realmente importa, de

manera simple, sin excederse en refinamientos y optimizaciones innecesarias. Si funciona así, déjelo así, si se va a perfeccionar u optimizar una rutina o programa se debe evaluar minuciosamente el costo beneficio.

XI. Las mejores arquitecturas, requerimientos y diseños emergen de los equipos auto-organizados. Los principios que rijan en equipo de trabajo deben surgir de su interior, los ajustes, estructuras administrativas deben formularse con la participación de todo el equipo teniendo siempre presente el bien colectivo, la responsabilidad es de todos.

XII. En intervalos regulares, el equipo reflexiona sobre cómo volverse más efectivo, entonces afina y ajusta su comportamiento como corresponde. El equipo de trabajo está todo el tiempo dispuesto a cambiar lo que sea necesario para mejorar. En cada tarea siempre existe la posibilidad de hacerlo mejor la próxima vez.

5.2. Implementaciones del manifiesto ágil

Existen en el mercado varias propuestas sobre cómo implementar los valores y principios enunciados, a las cuales los autores les colocan su toque personal dependiendo de su experiencia o necesidad.

Entre las más reconocidas se cuentan: **SCRUM**, **Crystal Methodologies, Dynamic Systems Development Method** (**DSDM**), **Adaptive Software Development** (**ASD**), **Feature-Driven Development** (**FDD**), **Lean Development** (**LD**) [2]

6. CONCLUSIONES Y RECOMENDACIONES

Las metodologías ágiles son una alternativa interesante para superar las debilidades de las metodologías convencionales, pero, al igual que los computadores no son en sí mismos la solución a los problemas de procesamiento de información, éstas no son la solución a todos los problemas que enfrenta el desarrollo de software.

Con el surgimiento de las metodologías ágiles, el concepto de etapa se desvanece dando paso a la idea de actividades, las cuales pueden ser organizadas a comodidad del equipo de trabajo, en paquetes pequeños conservando las mismas labores e identidad de las etapas concebidas en las primeras metodologías.

Es claro que la informática y la computación son ciencias nuevas, sin embargo ya es tiempo de que se les concedan espacios diversos y espontáneos en los que las personas que participan en los proyectos del área puedan sentirse más productivos con base en las circunstancias particulares de cada proyecto.

La participación directa del usuario durante todo el proyecto es una garantía de éxito, ya que la detección de los errores es más oportuna y de igual manera su corrección

Si bien los proponentes presentan su experiencia como muestra de conveniencia en el uso de sus metodologías, en términos generales éstas ofrecen mayores probabilidades de éxito en proyectos que cumplan ciertas condiciones como son no involucrar a más de 15 o 20 programadores, que no requieran más de cinco o seis meses de trabajo, que el entorno de desarrollo sea apropiado para la comunicación entre los miembros del equipo, que el cliente o usuario este dispuesto y disponible para el trabajo en equipo con el personal técnico y que los requerimientos puedan ser formulados según va avanzando el proyecto.

7. BIBLIOGRAFÍA

[1] PRESSMAN, Roger S. Ingeniería del Software: Un enfoque práctico, Quinta edición, 601 páginas, McGraw-Hill, Madrid 2002.

[2] LAWRECE, Shari P. Ingeniería del Software, Teoría y práctica, Primera edición, 759 páginas, Prentice Hall, Buenos Aires 2002.

[3] McCONNELL, Steve. Desarrollo y gestión de Proyectos informáticos, Primera edición, 691 páginas, McGraw-Hill, México 1997.

[4] KENDALL, Kenneth. Análisis y diseño de sistemas, Tercera edición, 913 páginas, Prentice Hall, México, 1997

[5] CRYSTAL, Agile project management. http://crystalmethodologies.org/. Visitado en Marzo 2006

[6] DSDM, Dynamic Systems Development Method. http://en.wikipedia.org/wiki/Dynamic_Systems_Development_Method#Principles_of_DSDM. Visitado en Marzo 2006

[7] FDD. Feature Driven Development http://en.wikipedia.org/wiki/Feature_Driven_Development. Visitado en Marzo 2006

[8] Historia de la Informática. http://ciberhabitat.gob.mx/museo/historia/texto/texto_guerra.htm. Visitado en Marzo 2006.

[9] Manifesto for Agile Software Development. http://agilemanifesto.org/ Visitado en Marzo 2006

[10] Máquina Analítica. http://www.dma.eui.upm.es/historia_informatica/Doc/Maquinas/MaqAnaliticaBabbage.htm. Visitado en Marzo 2006

[11] XP. Extreme programming. www.extremeprogramming.org._Visitado en Marzo 2006

[2] Para detalles sobre las diferentes propuestas de metodologías ágiles ver la bibliografía.