Evolución de las Metodologías de Desarrollo de la Ingeniería de Software en el Proceso la Ingeniería de Sistemas Software y Determinación de una metodología adaptable orientada a una organización pequeña

Evolution of Development Methodologies in Software Engineering Process Engineering of Software Systems and determining a customizable methodology aimed at a small organization

Autores: Luis Miguel Egas, Juan Xavier Játiva
Coautor: Ing. Lucas Garcés

Resumen

Desde el principio del uso de los ordenadores, al trabajar sobre el desarrollo de los primeros programas, se siguieron una serie de pautas o métodos para llevar a buen fin el proyecto. Bien es verdad que en esta situación la metodología era simple, era el típico proceso de abajo a arriba, con análisis insuficientes, ya que el problema era comprendido fácilmente en su totalidad. En la década de los setenta empezó a tomar cuerpo la idea de que si bien los procesos son importantes, y una incorrección en su tratamiento puede causar notables problemas e incomodidades, más importantes son los datos. Los procesos son, similares en todas las organizaciones y, en algunos casos, son hasta relativamente fácil de transportar. Sin embargo los datos son algo propio de la organización, algo totalmente diferente de los de las demás organización, y que la caracterizan. La década de los ochenta es la época marcada por las metodologías dirigida a datos cuya importancia va tomando cuerpo en las organizaciones. A mediados de la década el estado de la técnica permite a considerar entidades más complejas y con personalidad más acusada. Se empiezan a estudiar los objetos en sí como unidades de información. Los nuevos métodos van buscando minimizar riesgos y, puesto que los errores más perjudiciales se producen en los primeros pasos, se comienza ya desde la fase más general del estudio por analizar los riesgos que significa seguir con las siguientes fases del desarrollo. The Agile Alliance, una organización, sin ánimo de lucro, dedicada a promover loa conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que adopten dichos conceptos. El punto de partida fue el Manifiesto Ágil, un documento que resume la filosofía "ágil".1 A través del manifiesto surgen las Metodologías Ágiles, las cuales a continuación mencionaremos las características que vinculan con la realidad del Departamento TIC, acoplándose ágilmente al mencionar situaciones puntuales como la optimización de recursos, tiempos, costos y básicamente, la orientación a grupos multidisciplinarios.

Palabras Claves: *Proceso, ingeniería, software, ciclo de vida, métodos.*

Abstract

From the beginning of the use of computers, working on the development of the first programs, a set of guidelines or methods are followed to bring the project to fruition. It is true that in this situation the methodology was simple, it was a typical bottom-up process, with insufficient analysis since the problem was easily understood in its entirety. In the seventies began to take shape the idea that although the processes are important, and misconduct in its treatment can cause significant problems and discomfort are the most important data. The processes are similar in all organizations and, in some cases, are relatively easy to carry. However, the data are somewhat typical of the organization, something totally different from those of the other organization, and characterize it. The eighties is the time marked by methods directed data whose importance is taking shape in organizations. A mid the prior art allows to consider more complex and strong personality entities. Begin to study the objects themselves as units of information. The new methods are seeking to minimize risks and, since the most damaging errors occur in the first steps, and starts from the most general phase of the study to analyze the risks that go with the next phases of development. The Agile Alliance, an organization, a non-profit dedicated to promoting loa concepts related to agile software development and helping organizations to adopt these concepts. The starting point was the Agile Manifesto, a document summarizing the "agile" philosophy.
Through the manifesto Agile Methodologies arise, which then mention the characteristics that link with the reality of ICT Department, swiftly engaging the mention specific situations such as optimizing resources, time, cost and basically oriented multidisciplinary groups.

Keywords*: Process, engineering, software, life cycle, methods*

[1] Las Metodologías de desarrollo ágil como una oportunidad para la ingeniería del software educativo, (2008)

.

1. Introducción

Desde inicios de 1940, escribir software ha evolucionado hasta convertirse en una profesión que se ocupa de cómo crear software y maximizar su calidad.

La Ingeniería de software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software, y el estudio de estos enfoques, es decir, la aplicación de la ingeniería al software.

El proceso de la ingeniería de software requiere llevar a cabo numerosas tareas agrupadas en etapas, al conjunto de estas etapas se le denomina ciclo de vida. Las etapas comunes a casi todos los modelos de ciclo de vida son: análisis de requisitos, especificación, arquitectura, programación, prueba, documentación y mantenimiento.

Un objetivo de décadas ha sido el encontrar metodologías, modelos, paradigmas y filosofías de desarrollo, que sean sistemáticas, predecibles y repetibles, a fin de mejorar la productividad en el desarrollo y la calidad del producto software. Producto de esto el surgimiento de las metodologías tradicionales, de desarrollo rápido y ágiles.

2. Evolución Cronológica de las Metodologías de Desarrollo de Software

El desarrollo del software ha evolucionado, desde las primeras prácticas de desarrollo, pasando por el modelo de procesos, la conceptualización de los modelos del ciclo de vida del software, la propuesta de los modelos de la ingeniería del software, hasta desembocar en la metodología misma de los procesos de la ingeniería de software actuales.

2.1. Primera Etapa Cronológica (1950-1960): "Programación o técnicas de codificación"

En los años 50 el desarrollo estaba a cargo de programadores, por lo que se vio la importancia del análisis y diseño en el desarrollo de los sistemas. Aparecen los analistas programadores y analistas de sistemas [1].

En esta misma época, no existían metodologías de desarrollo. Las personas que desarrollaban los sistemas eran programadores más enfocados en la tarea de codificar, que en la de recoger y comprender las necesidades de los usuarios.

A medida que la complejidad de las tareas que realizaban las computadoras aumentaba, se hizo necesario disponer de un método sencillo para programar. Entonces, se crearon los lenguajes de programación de tercera generación [2] que diferían de las generaciones anteriores (lenguaje ensamblador conocido como segunda generación y lenguaje de máquina como primera generación) en que sus instrucciones o primitivas eran de alto nivel (comprensibles por el programador, como si fueran lenguajes naturales) e independientes de la máquina. Estos lenguajes se llamaron lenguajes de alto nivel

2.2. Segunda Etapa Cronológica (1960-1970): "Modelo de procesos"

El **modelo de procesos** predominaba para los años 60 y consistía en codificar y corregir (Code-and-Fix). Entonces, Codificar y corregir surge así como un modelo poco útil, pero sin embargo es la respuesta para muchos programadores al carecer de una estructura formal a seguir Si al terminar se descubría que el diseño era incorrecto, la solución era desecharlo y volver a empezar [3].

Paralelamente, aparece la **Crisis del Software**, llamada así por los problemas que surgieron a medida que se daba el desarrollo del software.

Esto provocó grandes pérdidas en la década de los 70´s sobre el desarrollo de software, dando como resultado una nueva disciplina llamada "Ingeniería del Software" que abarcaría los aspectos técnicos del software y la gestión de datos. (RUBI, 2012).

2.3. Tercera Etapa Cronológica (1970-1985): "Proceso de Desarrollo Software y Modelos Tradicionales del Ciclo de Vida"

Esta tercera etapa está marcada por las soluciones que deben realizarse para resolver la llamada "Crisis del Software". Mientras que el término **Ingeniería del Software** fue acuñado en la Conferencia de Ingeniería de Software de la OTAN en 1968 para discutir la crisis, los problemas que intentaba tratar empezaron mucho antes. Se volvió claro que los enfoques individuales al desarrollo de programas no escalaban hacia los grandes y complejos sistemas de software.

El **ciclo de vida de desarrollo de software** o SDLC (Software Develop Life Cicle) empezó a aparecer, a mediados de la década, como un consenso para la construcción centralizada de software, y daría las pautas en la que se logra establecer, desde que nace a partir de una necesidad, hasta que muere.

Para formalizar la estructura del ciclo de vida se logra establecer el **proceso de desarrollo software**, como una ayuda al proceso de resolución a los problemas, intentando transformar la necesidad en una solución automatizada que satisface la misma.

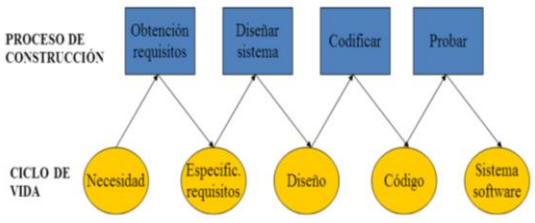

Figura 1. Figura Proceso y Ciclo de Vida

Es así, como la ingeniería del software se comienza a implantar y a valerse de una serie de modelos que establecen y muestran las distintas etapas y estados por los que pasa un producto software, desde su concepción inicial, pasando por su desarrollo, puesta en marcha y posterior mantenimiento, hasta la retirada del producto, a estos modelos se les denomina "**Modelos de ciclo de vida del software**".

El primer modelo publicado sobre el proceso de desarrollo software se derivó a partir de procesos más generales de ingeniería de sistemas. Debido al paso de una fase en cascada a otra, el modelo se conoce como **Modelo en Cascada**; definido por Winston Royce, el cual comenzó a diseñarse en 1966 y fue terminado alrededor de 1970 como respuesta al modelo de procesos. Este modelo sugiere un enfoque sistemático y secuencial para el desarrollo del software. Tiene más disciplina y se basa en el análisis, diseño, pruebas y mantenimientos [4]. Se define como una secuencia de fases, que en la etapa final de cada una de ellas se reúne la documentación para garantizar que cumple con las especificaciones y los requisitos antes de pasar a la siguiente fase. La importancia de éste modelo para la época fue en convertirse en un ejemplo de un proceso dirigido por un plan; en principio, se planificaría y programaría todas las actividades del proceso, antes de comenzar a trabajar con ellas

Posteriormente surgiría el **Modelo en V** propuesto por Alan Davis, desarrollado para terminar con algunos de los problemas que se vieron utilizando el enfoque de cascada tradicional. Los defectos estaban siendo encontrados demasiado tarde en el ciclo de vida, ya que las pruebas no se introducían hasta el final del proyecto. El modelo en V permitió hacer más explícita la tarea de la iteración de las actividades del proceso. Las pruebas que se implementarían en cada fase ayudarían a corregir posibles errores sin tener que esperar a que sean rectificados en la etapa final del proceso. Esto, sumado las pruebas unitarias y de integración se consigue obtener exactitud en los programas. Este modelo proporcionó un avance significativo al desarrollo software. A pesar de estar relacionado con el modelo de cascada, el modelo V demostró ser más efectivo ya que pudo con las pruebas que se realizarían durante cada fase, se detectó menos errores por que estos se pudieron corregir al momento, lo contradictorio de estas pruebas es que generaron muchos costos y también se perdía tiempo para la entrega al usuario final.

Paralelamente derivaría, como respuesta de las debilidades del modelo en cascada, el **Modelo Iterativo**. Este modelo buscaría un mejor desempeño del desarrollo software al reducir el riesgo que surge entre las necesidades del usuario y el producto final por malos entendidos durante la etapa de recogida de requisitos. Propondría en la **iteración** de varios ciclos de vida en cascada que al final de cada iteración se le entregaría al cliente una versión mejorada o con mayores funcionalidades del producto. Por lo tanto, el cliente es quien después de cada iteración evalúa el producto y lo corrige o propone mejoras. Estas iteraciones se repetirían hasta obtener un producto que satisfaga las necesidades del cliente.

Para inicios de 1980, fue propuesto el **Modelo de Desarrollo Incremental** por Harlan Mills, el cual combinaría elementos del modelo en cascada con la filosofía interactiva de construcción de prototipos. Surgió el enfoque incremental de desarrollo como una forma de reducir la repetición del trabajo en el proceso de desarrollo y dar oportunidad de retrasar la toma de decisiones en los requisitos hasta adquirir experiencia con el sistema. Se basaría en la filosofía de construir incrementando las funcionalidades del programa y aplicaría secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia lineal produce un incremento del software.

Tanto el Modelo Iterativo e Incremental dieron un avance significativo al desarrollo del software, pues permitieron la resolución de problemas de alto riesgo en tiempos tempranos del proyecto, una visión de avance en el desarrollo desde las etapas iniciales, una obtención del feedback del usuario lo antes posible, el aprendizaje y experiencia del equipo iteración tras iteración, una menor tasa de fallo del proyecto, mejor productividad del equipo, y menor cantidad de defectos, según demuestran estudios realizados sobre proyectos que han aplicado esta técnica. Pero a su vez presentan problemas agudos para sistemas grandes, complejos y de larga duración, y donde además diversos equipos desarrollan diferentes partes del sistema [5].

A finales de esta etapa, aparece el **desarrollo en espiral** de Barry Boehm en 1985 [6], el cual sería utilizado de forma generalizada en la ingeniería del software. A diferencia de otros modelos de proceso que finalizan cuando se entrega el software, el modelo se adaptaría para aplicarse a lo largo de toda la vida del software de cómputo. El nuevo producto evoluciona a través de cierto número de iteraciones alrededor de la espiral, teniendo un carácter permanente operativo hasta que el software se retira. La espiral presentaría un enfoque realista para el desarrollo de sistemas software a gran escala; considerándose así, el remedio de los modelos anteriores.

2.4. Cuarta Etapa Cronológica (1985-1999): "Métodos Rápidos e inicios del Desarrollo Ágil de la Ingeniería de Software"

A mediados de los años 80, es la época marcada por la distinción y conceptualización de los métodos de la ingeniería de software cuya importancia va tomando cuerpo en las organizaciones. Se empiezan a estudiar los objetos en sí como unidades de información, dando como punta pie inicial para el proceso evolutivo de desarrollo software el **modelo en espiral** presenciado a finales de la etapa anterior. Boehm, autor de diversos artículos de ingeniería del software; modelos de estimación de esfuerzos y tiempo que se consume en

hacer productos software; y modelos de ciclo de vida, ideó y promulgó un modelo desde un enfoque distinto al tradicional en Cascada: el Modelo Evolutivo Espiral.

Para los años 90 se quiere dar respuesta al entorno siempre cambiante y en rápida evolución en que se han de desarrollar los programas informáticos, lo cual da lugar a trabajar en ciclos cortos (como mini-proyectos) que implementan una parte de las funcionalidades, pero sin perder el rumbo general del proyecto global.

Surge entonces el desarrollo de software de "**métodos rápidos**" [7] (también denominado Modelo rápido o abreviado AG) los cuales reducirían el tiempo del ciclo de vida del software (por lo tanto, acelera el desarrollo) al desarrollar, en primera instancia, una versión prototipo y después integrar la funcionalidad de manera iterativa para satisfacer los requisitos del cliente y controlar todo el ciclo de desarrollo. Los métodos rápidos se originaron por la inestabilidad del entorno técnico y el hecho de que el cliente a veces es incapaz de definir cada uno de los requisitos al inicio del proyecto. El término "rápido" es una referencia a la capacidad de adaptarse a los cambios de contexto y a los cambios de especificaciones que ocurren durante el proceso de desarrollo.

Empezando con las ideas de Brian Gallagher, Alex Balchin, Barry Boehm y Scott Shultz, James Martin desarrolló el enfoque de desarrollo rápido de aplicaciones durante los 80 en IBM y lo formalizó finalmente en 1991, con la publicación del libro, "Desarrollo rápido de aplicaciones". Aparece entonces el **Desarrollo rápido de aplicaciones (RAD)**, para responder a la necesidad de entregar sistemas muy rápido. El método tiene una lista de tareas y una estructura de desglose de trabajo diseñada para la rapidez; comprende el desarrollo iterativo, la construcción de prototipos y el uso de utilidades CASE (Computer Aided Software Engineering). Tradicionalmente, el desarrollo rápido de aplicaciones tendría la tendencia a englobar también la usabilidad, utilidad y rapidez de ejecución. El desarrollo rápido de aplicaciones fue una respuesta a los procesos no ágiles de desarrollo desarrollados en los 70 y 80, tales como el método de análisis y diseño de sistemas estructurados y otros modelos en cascada. El enfoque de RAD no es apropiado para todos los proyectos. El alcance, el tamaño y las circunstancias, todo ello determinaría el éxito de un enfoque RAD.

En 1994, nace el **Método de desarrollo de sistemas dinámicos (DSDM),** desarrollado como un proceso de entrenamiento de negocios en Inglaterra, se estableció para crear una metodología RAD unificada, el cual definiría el marco para desarrollar un proceso de producción de software. El aporte como metodología radica en entregar sistemas software en tiempo y presupuesto ajustándose a los cambios de requisitos durante el proceso de desarrollo software, comprometiendo a la participación continua con el usuario como clave principal para llevar a cabo un proyecto eficiente y efectivo, de tal forma que las decisiones se tomarían de forma más exacta.

En 1995 Schwaber y Sutherland, durante el OOPSLA '95 [8], presentaron en paralelo una serie de artículos describiendo **Scrum**, siendo ésta la primera aparición pública del método. Durante los años siguientes, Schwaber y Sutherland, colaboraron para consolidar los artículos antes mencionados, así con sus experiencias y el conjunto de mejores prácticas de la industria que conformarían a lo que actualmente se le conoce como Scrum. Para 2001, Schwaber y Mike Beedle describirían la metodología en el libro Agile Software Development with Scrum. Este método definiría un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Ha contribuido como método a la generación de software de calidad, principalmente por apoyar a proyectos con un rápido cambio de requisitos y al presentar una guía para las actividades de desarrollo dentro de un proceso de análisis incorporando actividades estructurales como: requerimientos, análisis, diseño, evolución y entrega; demostrando así, ser eficaz para proyectos con plazos de entrega muy apretados, requerimientos cambiante y negocios críticos.

Para 1996, surge **Extreme Programming** (**XP**) o Programación Extrema fundada por Ken Beck, identificando qué era lo simple y difícil al momento de programar. Centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP ha contribuido como método a la generación efectiva de software, fundamentada en la realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP promueve una adecuada práctica para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

En junio del 1998 se lanza **Rational Unified Process RUP** (Proceso unificado racional), siendo un método de desarrollo iterativo promovido por la compañía Rational Software, que fue comprada por IBM. El método RUP especifica, principalmente, la constitución del equipo y las escalas de tiempo, así como un número de modelos de documento. El enfoque práctico de RUP describe las buenas prácticas de ingeniería de software que se recomiendan para su uso en desarrollo de sistemas software.

2.5. Quinta Etapa Cronológica (2000 al presente): "Metodologías del Proceso de la Ingeniería de Software"

Con la creciente demanda de software en muchas organizaciones pequeñas, la necesidad de soluciones de software de bajo costo llevó al crecimiento de métodos más simples y rápidos que desarrollaran software funcional, de los requisitos de

implementación, más rápidos y más fáciles [9]. El uso de prototipos rápidos, procedimientos y normas evolucionó al uso del concepto de la Metodología del Proceso de la Ingeniería de Software, destacando la utilización de las metodologías ligeras completas como la programación extrema (XP), que intentó simplificar muchas las áreas de la ingeniería de software, incluyendo la recopilación de requerimientos y las pruebas de confiabilidad para el creciente y gran número de pequeños sistemas de software. Sistemas de software muy grandes todavía utilizan metodologías muy documentadas, con muchos volúmenes en el conjunto de documentación; sin embargo, sistemas más pequeños tenían un enfoque alternativo más simple y rápido para administrar el desarrollo y mantenimiento de cálculos y algoritmos de software, almacenamiento y recuperación de información y visualización.

En el año 2001, miembros prominentes de la comunidad de desarrollo software se reunieron en Snowbird, Utah, y adoptaron el nombre de "**métodos ágiles**". Poco después, algunas de estas personas formaron la "alianza ágil", una organización sin fines de lucro que promueve el desarrollo ágil de aplicaciones. El desarrollo ágil de software guiaría a los proyectos de desarrollo de software que evolucionan rápidamente con cambiantes expectativas y mercados competitivos. Los proponentes de este método creen que procesos pesados, dirigidos por documentos (como TickIT, CMM e ISO 9000) están desapareciendo en importancia. Algunas personas creen que las empresas y agencias exportan muchos de los puestos de trabajo que pueden ser guiados por procesos pesados. Es así, como se introduce el manejo de las **Metodologías de Desarrollo Ágil**, contemplando un conjunto de métodos de ingeniería del software, que se basan en el desarrollo iterativo e incremental, teniendo presente los cambios y respondiendo a estos mediante la colaboración de un grupo de desarrolladores auto-organizados y multidisciplinares.

Muchos métodos similares al ágil fueron creados antes del 2000. Entre los más notables se encuentran: Scrum, Crystal Clear (cristal transparente), programación extrema (en inglés eXtreme Programming o XP), desarrollo de software adaptativo (ASD), feature driven development (FDD), Método de desarrollo de sistemas dinámicos (en inglés Dynamic Systems Development Methodo DSDM).

En las metodologías ágiles, los procesos se desarrollan de manera solapada, donde el ciclo de vida del proyecto, es cíclico. La diferencia en el ciclo de vida de un proyecto ágil, en comparación con uno tradicional, se debe a la forma en la que el agilismo, solapa los procesos de manera iterativa.

Se puede apreciar claramente la evolución paulatina que ha desembocado hasta conformar la concepción de las **Metodologías del Proceso de Ingeniería de Software**, la cual describe el conjunto de herramientas, técnicas, procedimientos y soporte documental para el diseño del **Sistema Software**.

Por otro lado, Sommerville (2002) define que "una metodología de ingeniería de software es un enfoque estructurado para el desarrollo de software cuyo propósito es facilitar la producción de software de alta calidad de una forma costeable", cabe destacar que para usar este enfoque se debe manejar conceptos fundamentales tales como; procesos, métodos, tareas, procedimientos, técnicas, herramientas, productos, entre otros.

Esta etapa cronológica logra establecer metodologías que pueden involucrar prácticas tanto de **Metodologías Ágiles** como de **Metodologías Tradicionales o Convencionales**. Tener metodologías diferentes para aplicar de acuerdo con el proyecto que se desarrolle resulta una idea interesante. De esta manera podríamos tener una metodología para cada proyecto, la problemática sería definir cada una de las prácticas, y en el momento preciso definir parámetros para saber cual usar. Es importante tener en cuenta que el uso de un método ágil no es para todos. Sin embargo, una de las principales ventajas de los métodos ágiles es su peso inicialmente ligero y por eso las personas que no estén acostumbradas a seguir procesos encuentran estas metodologías bastante agradables.

Los nuevos métodos que paulatinamente van generándose, buscan minimizar riesgos y, puesto que los errores más perjudiciales se producen en los primeros pasos, se comienza ya desde la fase más general del estudio por analizar los riesgos que significa seguir con las siguientes fases del desarrollo. Si los riesgos son superiores a lo que se consideran permisibles, se vuelve al paso anterior o se abandona el desarrollo. No sólo se realizan desarrollos lineales, en cascada, sino también desarrollos y métodos en espiral que son notablemente más complejos, apoyándose también con el desarrollo ágil.

3. Resultados

En los inicios de la programación, no existía una metodología, método o técnica específica que permita el desarrollo de software, lo que ocasionaba insatisfacción en los usuarios. Surge entonces un modelo de procesos consistido en codificar y corregir siendo esta una respuesta inicial a la necesidad de una estructura de programación. En los 70, los elevados costos, tiempos excesivos, insatisfacción del usuario, etc., dan lugar a la Crisis del Software. Aparece una nueva disciplina "La Ingeniería de Software" que abarcaría los aspectos técnicos del software y la gestión de datos. Se comienza a hablar del ciclo de vida del software logrando establecer los estados por los que pasa el producto software desde que nace a partir de una necesidad, hasta que muere. Esta estructura logra establecer el Proceso de Desarrollo Software. Posteriormente, se formalizarían los Modelos del Ciclo

de Vida del Software tradicionales, los cuales pretenden abarcar todo el proceso completo creando en cada paso normativas y parámetros describiendo todo el proceso de desarrollo. A mediados de los 80, se comienza a manejar el concepto de los Métodos de Desarrollo Software con métodos iterativos, evolutivos y rápidos. A finales de los 90, se empieza a manejar métodos de desarrollo ágil de software, el cual guiaría a los proyectos software que evolucionan rápidamente con cambiantes expectativas y mercados competitivos como XP, Scrum, ASD, FDD, etc. Todo el proceso evolutivo que se presenció en este artículo, logra formalizar, hasta la actualidad, a las Metodologías del Proceso de la Ingeniería de Software, las cuales pueden describirse como el conjunto de herramientas, técnicas, procedimientos y soporte documental para el diseño del Sistema Software.

Selección de las Metodologías a ser investigadas.

4. El Manifiesto Ágil.

Como ya se había argumentado en la introducción, a principio del año 2001, luego de celebrarle una reunión en Utah-EEUU, nace el término ágil, aplicado al desarrollo de software.

Participaron en esta reunión un seleccionado grupo de 17 expertos de la industria del software, incluyendo algunos de los creadores o impulsores de metodologías del software. Con un objetivo específico de esbozar los valores y principios que posibilitaran a los equipos desarrollar software rápidamente y respondiendo a su vez, a los cambios que puedan surgir a lo largo del proyecto que se estuviera desarrollando.

En esta reunión se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, debido a que estos se caracterizaban por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades que se desarrollaban.

Luego de desarrollarse esta reunión se creó "The Agile Alliance"2, una organización fundada sin ánimos de lucro, que se dedicaba a promover conceptos relacionados con el desarrollo ágil de software y también a impulsar dichas organizaciones para que adopten dichos conceptos. Donde se utilizó como punto de partida el Manifiesto Ágil, documento que resume la filosofía ágil.

"Según este Manifiesto, se valora:

Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas: Considerando a la gente como el principal factor de éxito de un proyecto software.

Como primicia se destaca que es más importante construir un buen equipo que construir el entorno. Muchas veces se comete el error de hacer lo contrario; se construye primero el entorno y se espera que el equipo se adapte automáticamente. Evidentemente, es satisfactorio crear el equipo primero y que luego éste configure su propio entorno de desarrollo en base a sus necesidades.

Desarrollar software que funciona más que conseguir una buena documentación: Se impone como regla general a seguir, no producir documentos a menos que sean necesarios de forma inmediata para tomar una decisión importante. Acentuar que estos documentos deben ser cortos y centrarse en lo fundamental.

4.1. Estudio de las metodologías ágiles más relevantes.

A fin de garantizar el estudio selectivo y que sea válido y justificable, se dispondrá de la información necesaria de las metodologías con las que se trabajará y que fuesen lo suficientemente relevantes.

El objetivo de este apartado es seleccionar un total de cinco metodologías ágiles, que posteriormente serán caracterizadas y sobre las cuales realizaremos seguidamente una comparativa. Las metodologías que hemos considerado a la hora de realizar la selección han sido:

1. Adaptative Software Development
2. Agile Modeling
3. Agile Model Driven Development
4. Agile Project Management
5. Agile Unified Process
6. Crystal Methods
7. Dynamic Systems development methods
8. Feature driven development
9. Internet Speed Development
10. Lean development
11. Pragmatic programming
12. Scrum
13. Test Driven Development
14. XBreed
15. Extreme Programming
16. WinWinSpiral.

Todas estas metodologías están documentadas o referenciadas como metodologías ágiles en las siguientes referencias [56][57][58][59][60][61].

No son todas las metodologías ágiles que existen, pero sí son una gran representación de ellas. A continuación

[2]www.agilealliance.com

mostramos algunas metodologías ágiles que no hemos incluido en la preselección:

1. Evolutionary Project Management.
2. Story cards driven development.
3. Agile UnifiedProcess
4. Open UnifiedProcess

Tal y como hemos mencionado anteriormente, nos interesa trabajar con metodologías lo suficientemente documentadas, que nos faciliten la obtención de información, pero también es interesante trabajar con metodologías que dispongan de algún tipo de certificación y training. Según estas condiciones hemos determinado seis clasificaciones donde escogeremos a las seis metodologías que se encuentran mejor posicionadas; estas son las clasificaciones:

- La metodología con mayor presencia en Internet.
- La metodología mejor documentada.
- Metodologías certificadas y con training.
- Metodologías con comunidades.
- Metodología más utilizada por empresas. Presencia empresarial.
- Metodología más utilizada en proyectos software.

4 .1.1 La metodología con más presencia en internet.

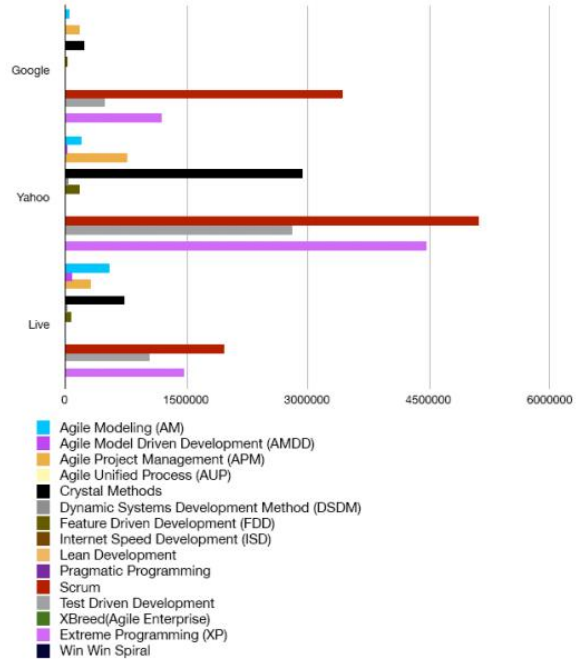

Figura 1: Mayor Presencia en Internet de las Metodologías Ágiles

Fuente: José Carvajal, Metodologías Ágiles[55]

Según el número de resultados obtenidos en las búsquedas por Yahoo, Google y Microsoft Live, las cinco metodologías con mayor presencia en la red y en este orden son (ver Figura 1):

1. Scrum
2. Extreme Programming (XP)
3. Feature Driven Development (FDD)
4. CrystalMethods
5. Adaptative Software Development (ASD)

4.1.2 La metodología mejor documentada.

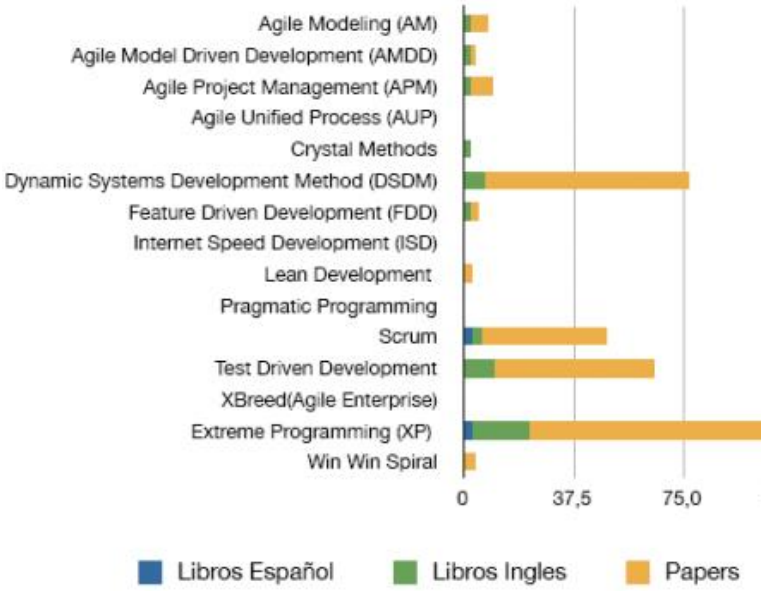

Figura 2: Mejor documentación de las Metodologías Ágiles
Fuente: José Carvajal, Metodologías Ágiles [55]

Las cinco metodologías mejor documentadas son (ver figura 2):

1. Extreme Programming (XP)
2. Feature Driven Development (FDD)
3. Dynamic System Development Method (DSDM)
4. Scrum
5. Adaptative Software Development (ASD)

4.1.3 Metodologías certificadas y con training.

Metodologías con certificación:

1. Dynamic System Development Method tieneun DSDM certified.
2. Feature Driven Development tiene FDD Certified.
3. Scrum dispone de ScrumCertified.

Metodologías con training:

1. Agile Modeling.
2. Agile Model Driven Development.
3. Adaptative Software Development.
4. Crystal methods.
5. Dynamic System Development Method.
6. Feature Driven Development.
7. Lean Development.
8. Scrum.
9. Extreme Programing.

4.1.4 Metodologías con comunidades.

La mayoría pertenecen a la Agile Alliance, pero algunas han montado auténticas comunidades y alianzas a su alrededor.

Metodologías asociadas a la Agile Alliance:

1. Agile Modeling.
2. Agile Model Driven Development.
3. Crystal methods.
4. Dynamic System Development Method.
5. Feature Driven Development.
6. Internet Speed Development.
7. Lean Development.
8. Pragmatic Programming.
9. Scrum.
10. Test Driven Development.
11. Extreme Programing.
12. WinWinSpiral.

- Metodologías con comunidades o alianzas diferentes:

- Adaptative Software Development, con Declaration of Interdependence for modern management.

- Dynamic System Development Method, con DSDM Consortium.

- Scrum, con Scrumalliance.

4.1.5 Metodología más utilizada por empresas. Como se pudo observar es realmente complicado encontrar ejemplos de proyectos realizados en una empresa privada y con una metodología en concreto.

4.1.6 Determinación de las características de cada una de las metodologías seleccionadas.

A continuación se resumen algunos métodos ágiles con sus definiciones, funciones y características, que están siendo utilizados con éxito en proyectos reales:

Adaptive Software Development (ASD): Tiene como principales características ser: iterativo, orientado a los componentes software más que a las tareas y tolerante a los cambios. El ciclo de vida que propone tiene tres fases esenciales: especulación, colaboración y aprendizaje.

Especulación, colaboración y aprendizaje.

Crystal Methods: El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas.

Crystal no especifica ningún ciclo de vida concreto[21], ni ningún modelo de procesos, en vez de eso lo que hace es determinar una guía de las políticas estándar, productos de trabajo, asuntos locales, herramientas, estándares y roles.

Método de desarrollo de sistemas dinámicos (DSDM): Define el marco para desarrollar un proceso de producción de software.
Propone cuatro fases: Estudio viabilidad, estudio del negocio, modelado funcional, diseño, construcción, e implementación.

Scrum: Define un marco para la gestión de proyectos, está especialmente indicada para proyectos con un rápido cambio de requisitos. Su ciclo de vida es: Planeamiento, montaje, desarrollo y liberación [22].

Programación Extrema (XP): Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, con el siguiente ciclo de vida [23]: exploración, planificación de la Entrega, iteraciones, producción, mantenimiento y muerte del Proyecto.

5. Estudio comparativo de las metodologías más relevantes a fin de determinar la metodología que satisface las necesidades del Departamentos TIC's de organizaciones pequeña

5.1. Capacidad de agilidad.

Representa cuál es la parte ágil de la metodología. Los atributos de esta vista representan todos los aspectos del concepto de agilidad y su evaluación refleja que aspectos están incluidos en una metodología.

Una metodología de desarrollo de software está compuesta por un ciclo de vida. En Ingeniería del Software existen diferentes ciclos de vida, como por ejemplo modelo en V, modelo en espiral, etc. De acuerdo a la cronología de aparición de las metodologías ágiles, la mayoría de las metodologías derivan directamente del modelo en espiral. Esto se explica ya que las dos principales características de modelo en espiral son un ciclo de vida iterativo e incremental. Por tanto, los cambios de requisitos se pueden ir integrando en cada iteración, de manera que el plan de trabajo no tiene que ser modificado, irá cambiando a lo largo de las iteraciones. Otro punto interesante es la duración de las iteraciones, con iteraciones cortas aumenta el número de reuniones con el cliente para definir y detallar sus necesidades

5.2. Aplicabilidad

El objetivo de esta vista es mostrar el impacto de los aspectos ambientales en el método. Representa cuando el entorno es favorable para la aplicación de metodologías ágiles. Este aspecto se describe por atributos, cada uno correspondiente a una característica del entorno.

5.3. Procesos y Productos

La vista de los procesos y productos representa cómo se caracteriza la metodología. Los atributos caracterizarán a los procesos ágiles por dos dimensiones y listarán los productos de las actividades del proceso.
El proceso se compone de dos dimensiones. La primera dimensión son las actividades de desarrollo de software cubiertas por las metodologías ágiles. La segunda representa el nivel de abstracción de sus directrices y reglas. Estas dos dimensiones se evalúan con atributos de esta vista.

5.4 Formulario Orientación tradicional vs Orientación ágil

ORIENTACIÓN ÁGL		ORIENTACIÓN TRADICIONAL	
VALOR	IMPORTANCIA	VALOR	IMPORTANCIA
Individuo y las interacciones del equipo	3	El proceso y las Herramientas	2
Desarrollar software que funciona	3	Conseguir una buena Documentación	2
Colaboración con el Cliente	2	Negociación Contractual	2
Respuesta al cambio	3	Seguimiento de un Plan	2
MEDIA	2,75		2

De acuerdo a los resultados arrojados, se demuestra que se sobrevalora lo indicado por los valores del manifiesto ágil, con una orientación ágil de media 2,75 mayor a una orientación tradicional de media 2.

5.5 Cumplimiento de principios ágiles

	Principios del Manifiesto Ágil	Prioridad
1	La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.	2
2	Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.	3
3	Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.	2
4	La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.	2
5	Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo	3
6	El diálogo cara a cara es el método más efectivo para comunicar información dentro de un equipo de desarrollo.	3

	Principios del Manifiesto Ágil	Prioridad
7	El software que funciona es la medida principal de progreso.	3
8	Los procesos ágiles promueven un desarrollo sostenible. Los promotores, los desarrolladores y usuarios deberían ser capaces de mantener una paz constante	3
9	La atención continua a la calidad técnica y al buen diseño mejora la agilidad.	3
10	La simplicidad es esencial.	3
11	Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.	2
12	En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.	2
	TOTAL	31/36

Según el resultado obtenido, se deduce que las metas según el enfoque del equipo de desarrollo del Departamento TIC se orientan en un 86,11% al cumplimiento íntegro o total de los principios ágiles.

5.7 Estado actual de las metodologías (grado de madurez)

Metodología	Estado Actual
ASD	En construcción
Crystal Methods	En construcción/ Activa
DSDM	Activa
SCRUM	Activa
FDD	Activa
XP	Activa

6. Determinación de la Metodología Ágil adaptable para el Departamentos TIC de organizaciones pequeñas.

Se ha realizado un estudio selectivo fuertemente válido y justificable, del que se dispuso de la información necesaria en base a las metodologías que se consideró más importantes.

Se partió desde la identificación de las Metodologías ágiles, evaluando los criterios válidos en base a información real existente, las que determinaron la selección de seis metodologías.

Luego de la selección se determinaron las características esenciales dando a conocer el proceso o ciclo de vida de las mismas.

Finalmente, se realizó un estudio comparativo de las metodologías seleccionadas en función de las necesidades, ágil o tradicional (necesaria para fundamentar a fondo el estudio), y una segunda parte que permitiría conocer la metodología ágil, con la ayuda del Framework propuesto por Adrian Iacovelli.

Además, apoyados con la comparativa del estado actual de las metodologías y en base a todos los resultados arrojados del estudio comparativo, podemos identificar que la metodología que más se adapta es la Extreme Programming (XP); recalcando que no es la única que podría ajustarse y que existen otras como la DSDM, ASP, Scrum (las cuales tuvieron ponderaciones altas) que incluso las podemos combinar.

7. Conclusiones

En el estudio del artículo, se puede determinar que no existe una metodología universal para hacer frente con éxito a cualquier proyecto de desarrollo de software. Toda metodología debe ser adaptada al contexto del proyecto: recursos técnicos y humano, tiempo de desarrollo, tipo de sistema, etc.

Históricamente, las metodologías tradicionales han intentado abordar la mayor cantidad de situaciones de contexto del proyecto, exigiendo un esfuerzo considerable para ser adaptadas, sobre todo en proyectos pequeños y con requisitos muy cambiantes.

Las metodologías ágiles ofrecen una solución casi a medida para una gran cantidad de proyectos que tienen estas características. Una de las cualidades más destacables en una metodología ágil es su sencillez, tanto en su aprendizaje como en su aplicación, reduciéndose así los costes de implantación en un equipo de desarrollo.

La industria del software ha enseñado a las empresas desarrolladoras, la adopción de una metodología de desarrollo en sus inicios, ya que a medida que crece la experiencia se va adoptando mejoras en el entrono de las metodologías de desarrollo software.

8. Agradecimientos

Un agradecimiento especial a la Universidad de Fuerzas Armadas ESPE Extensión Latacunga, cuna de futuros profesionales y emprendedores del país, por la formación académica-profesional recibida en sus aulas y al Ing. Lucas Garcés por su constante colaboración para el desarrollo del artículo.

9. Referencias

[1] Carballar, D. *Ingeniería de software* [documento en línea].
« www.eduinnova.es/dic09/Ingenieria_Software.pdf » [consulta: 10 de junio de 2012]

[2] Programación en Pascal, Introducción a las Computadoras y a los lenguajes de programación, Mc. Graw Hill, Pág 34.

[3] Hernán, M. (2004). *Diseño de una Metodología Ágil de Desarrollo de Software*. Tesis de Grado de Ingeniería en Informática. Universidad de Buenos Aires. Pág. 11-12.

[4] Sommerville, I. (2006). *Ingeniería del software* (Séptima Edición). Madrid. Pág. 62.

[5] Sommerville, I. (2011). *Ingeniería del software* (Novena Edición). México. Pág. 34.

[6] http://es.wikipedia.org/wiki/Metodolog%C3%ADa_de_desarrollo_de_software.

[7] http://es.kioskea.net/contents/227-metodos-rapidos-rad-xp.

[8](PDF) Rising, L., Janoff, N.S. (2000). The Scrum Software Development Process for Small Teams Retrieved March 15, 2007.

[9] http://es.wikipedia.org/wiki/Historia_de_la_ingenier%C3%ADa_del_software.

[10] Cota, A. 1994. Ingeniería de Software. Soluciones Avanzadas. Julio de 1994.

[11] Jacobson, I. 1998. Applying UML in the Unified Process. Presentación. Rational Software. Disponible en http://www.rational.com/uml como UMLconf.zip [Acceso: Diciembre 2009]

[12] Gruber, T.R. 1993. A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, ISSN 1042-8143, vol. 5, Nº 2, pp. 199-220.

[13] W3C. 2004. "OWL Web Ontology Language Use Cases and Requirements," W3C Recommendation 10 February 2004. Available: http://www.w3.org/TR/2004/REC-webont-req-20040210/

[14] Noy, N.F. and Musen, M.A. 2004. "Ontology versioning in an ontology management framework" Intelligent Systems, IEEE vol. 19, Nº 4, pp. 6-13, Jul-Aug 2004.

[15] Naur, Peter and Brian Randell (eds.). 1969. Software Engineering: Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October, 1968. Brussels, Scientific Affairs Division, NATO. See Naur et al. 1976.

[16] Pressman, R. 2005. Ingeniería del Software: Un enfoque práctico, Sexta Edición. McGraw-Hill Interamericana.

[17] Paulk, M. y colegas. 1993. Capability Maturity Model for Software. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

[18] Juzgado, J.N. 1996. Procesos de construcción del software y ciclos de vida. Universidad Politécnica de Madrid.

[19] Sigwart, C. et al. 1990. Software Engineering: a project oriented approach. Franklin, Beedle y Associates, Inc., Irvine, California.

[20] Gómez-Pérez, A., Fernández-López, M., and Corcho M. 2004. Ontological Engineering. Springer Verlag London

[21]Cristal Clear, http://www.scribd.com/doc/55555056/Crystal-Clear-Version-Open-Document

[22] Ciclo de vida XP, http://oness.sourceforge.net/proyecto/html/ch05s02.html

[23] Ciclo de vida Scrum, pag 20, http://carlosreynoso.com.ar/archivos/arquitectura/Metodos-Agiles.PDF