EVOLUCIÓN DE LAS METODOLOGÍAS DE DESARROLLO DE LA INGENIERÍA DE SOFTWARE EN EL PROCESO LA INGENIERÍA DE SISTEMAS SOFTWARE

Garcés Guayta Lucas Rogerio[1] Luis Miguel Egas[2]

1. Docente tiempo completo - Universidad de las Fuerzas Armadas ESPE Extensión Latacunga - Departamento de Eléctrica y Electrónica, lrgarces@espe.edu.ec
2. Egresado de la Carrera de Ingeniería de Software - Universidad de las Fuerzas Armadas ESPE Extensión Latacunga - Departamento de Eléctrica y Electrónica, Luisegas10@gmail.com

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. Por lo tanto en el principio de la informática los métodos eran de tipo ascendente y orientado a procesos. A finales de la década de los sesenta empezaron a aparecer ordenadores en las empresas para resolver problemas del tipo de cálculo de nómina, no de gestión de personal, debido a que era una labor que exige hacer muchos cálculos, en general repetitivos, y por tanto con alta probabilidad de error. 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.

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 their entirety . Therefore the principle of computer methods were bottom-up process-oriented . In the late sixties began appearing computers in business to solve problems like payroll , no personnel management , because it was a task that requires many calculations generally repetitive and therefore high probability of error. 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 the methodologies 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..

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

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 calidad puede referirse a cuán mantenible es el software, su estabilidad, velocidad, usabilidad, comprobabilidad, legibilidad, tamaño, costo, seguridad y número de fallas o "bugs", así como, entre muchos otros atributos, a cualidades menos medibles como elegancia, concisión y satisfacción del cliente. La mejor manera de crear software de alta calidad es un problema separado y controvertido cubriendo el diseño de software, principios para escribir código, llamados "mejores prácticas", así como cuestiones más amplias de gestión como tamaño óptimo del equipo de trabajo, el proceso, la mejor manera de entregar el software a tiempo y tan rápidamente como sea posible, la "cultura" del lugar de trabajo, prácticas de contratación y así sucesivamente.

Pero para poder hablar de calidad, se tuvo que generar un gran proceso histórico, que hasta hoy en día sigue en constante evolución, es el caso de las Metodologías de Desarrollo de Software. Estas proponen como objetivo principal presentar un conjunto de técnicas tradicionales, modernas y ágiles de modelado de sistemas que permitirían desarrollar software con calidad, incluyendo heurísticas de construcción y criterios de comparación de modelos de sistemas.

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.

Aunque se disponía de las más recientes técnicas de codificación de la época, los usuarios a menudo, no quedaban satisfechos con el sistema software que se generaba, porque sus necesidades no estaban definidas con claridad en una fase de análisis previo. Ante esta perspectiva se vio la importancia del análisis y del diseño en el desarrollo de un sistema.

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. Los ejemplos más conocidos son FORTRAN (FORmula TRANslator) que fue desarrollado para aplicaciones científicas y de ingeniería, y COBOL (COmmon Business-Oriented Language), que fue desarrollado por la U.S. Navy de Estados Unidos, para aplicaciones de gestión o administración.

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

De las técnicas de codificación de la primera etapa que desembocó en la inconformidad de los usuarios por la carencia de un análisis inicial, nace la necesidad de buscar alternativas para esquematizar de alguna manera la producción del software.

Desde que se empezó a trabajar sobre el desarrollo de programas, se siguieron ciertos métodos que permitían llevar a producir un buen proyecto, estas metodologías aplicadas eran simples, solo se preocupaban por los procesos mas no por los datos, por lo tanto los métodos eran desarrollados hacia los 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. Realmente se la consideró aplicable, ya que de hecho la programación se la llevaba de forma intuitiva. A pesar que el desarrollo de software tomó una connotación de tarea unipersonal y donde el programador era el usuario de la aplicación, se lo consideró como una base inicial para la fabricación del software, en vista de que en este modelo se empieza a establecer una idea general de lo que se necesita construir, se utiliza cualquier combinación de diseño, código, depuración y métodos de prueba no formales que se los aplica hasta que se tiene el producto listo para entregarlo. Si al terminar se descubría que el diseño era incorrecto, la solución era desecharlo y volver a empezar[3]. Este modelo implementaba el código y luego se pensaba en los requisitos, diseño, validación y mantenimiento.

Paralelamente, aparece la **Crisis del Software**, llamada así por los problemas que surgieron a medida que se daba el desarrollo del software. Especialmente fue marcada por los excesos de costos, la escasa fiabilidad, la insatisfacción de los usuarios y el tiempo de creación de software que no finalizaba en el plazo establecido; todos estos aspectos conocidos como "síntomas" de la crisis de 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. Éstos no eran confiables, costaban más de lo esperado y se distribuían con demora. La historia de la ingeniería del software está entrelazada con las historias contrapuestas de hardware y software.

Para dar soluciones que enfrentaba el software, a inicios de la década de los setenta, se empezó a tomar la importancia de los datos, y para solucionar sistemas complejos empezó el análisis por partes o etapas, se introducen la planeación y la administración.

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, de manera general, los estados por los que pasa el producto software 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. De esta manera, ambos conceptos se formarían para tratar de afrontar la crisis de la etapa anterior.

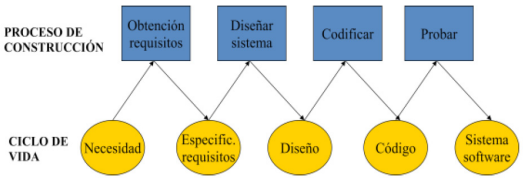

Figura 1. Figura Proceso y Ciclo de Vida

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"**, los cuales pretenden abarcar todo el proceso completo creando en cada paso normativas y parámetros describiendo el desarrollo de software desde la fase inicial hasta la final y orientarlo a ajustarse a las propias necesidades de cada empresa y que sean aceptados internacionalmente. El aporte de dichos modelos al desarrollo del software se focaliza en el suministro de una guía para los ingenieros de software con el fin de ordenar las diversas actividades técnicas en el proyecto, por otra parte suministran un marco para administrar el progreso de desarrollo y el mantenimiento, en el sentido en que permitirían estimar recursos, definir puntos de control intermedios, monitorear el avance, etc.

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. Entonces, comienza a enseñar a los programadores que el proceso de software no es un simple modelo lineal, sino que implica retroalimentación de una fase a otra. Por otro lado, se empieza a descubrir los errores y las omisiones en los requerimientos originales del software. Surgen los errores de programa y diseño, y se detecta la necesidad de nueva funcionalidad. Su principal problema detectado es la partición inflexible del proyecto en distintas etapas. Tienen que establecerse compromisos en una etapa temprana del proceso, por lo que dificulta responder a los requerimientos cambiantes del cliente. Por lo tanto, nace el sentido de evolucionar el sistema para mantenerse útil.

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. Su modelo de ciclo de vida en espiral tiene en cuenta fuertemente el riesgo que aparece a la hora de desarrollar software. Para ello, se comienza mirando las posibles alternativas de desarrollo, se opta por la de riesgos más asumibles y se hace un ciclo de la espiral. Si el cliente quiere seguir haciendo mejoras en el software, se vuelven a evaluar las nuevas alternativas y riesgos y se realiza otra vuelta de la espiral, así hasta que llegue un momento en el que el producto software desarrollado sea aceptado y no necesite seguir mejorándose con otro nuevo ciclo. Este modelo no fue el primero en tratar el desarrollo iterativo, pero fue el primer modelo en explicar las iteraciones. Este modelo, Boehm lo propone en 1988 en su artículo "A Spiral Model of Software Development and Enhancement" y básicamente consiste en una serie de ciclos que se repiten en forma de espiral, comenzando desde el centro. Se suele interpretar como que dentro de cada ciclo de la espiral se sigue un modelo en cascada, pero no necesariamente ha de ser así.

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.

La tendencia del control de procesos para desarrollo de software ha traído como resultado que proyectos no resulten exitosos debido al cambiante contexto que existe, por lo cual las metodologías ágiles pretenden resolver este inconveniente, construyendo soluciones a la medida asegurando la calidad. Los métodos ágiles fueron pensados especialmente para equipos de desarrollo pequeños, con plazos reducidos, requisitos volátiles y nuevas tecnologías. La filosofía de las metodologías ágiles, pretenden dar mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad.

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.

Particularmente, una metodología se basa en una combinación de los modelos de proceso genéricos para obtener como beneficio un software que solucione un problema. Adicionalmente una metodología debería definir con precisión los artefactos, roles y actividades, junto con prácticas, técnicas recomendadas y guías de adaptación de la metodología al proyecto. Sin embargo, la complejidad del proceso de creación de software es netamente dependiente de la naturaleza del proyecto mismo, por lo que el escogimiento de la metodología estará acorde al nivel de aporte del proyecto, ya sea pequeño, mediano o de gran nivel.

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 cuál 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. Por otro lado, las metodologías tradicionales o convencionales permiten crear software de manera más segura ya que estas están más establecidas según por sus pasos. Por otra parte las metodologías de desarrollo comienzan a interesarse no sólo en lograr que el proyecto sea puesto en funcionamiento sino en minimizar costos durante su desarrollo y sobre todo durante su mantenimiento.

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.

4.- 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.

Sigue siendo un reto para la industria del software la aplicación de metodologías ágiles en proyecto grandes.

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

5.- 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. Diseño de una Metodología Ágil de Desarrollo de Software. Tesis de Grado de Ingeniería en Informática. Universidad de Buenos Aires. Pág. 11-12. (2004).

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

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

[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. The Scrum Software Development Process for Small Teams Retrieved March 15, 2007. (2000).

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

[10] COTA, A. Ingeniería de Software. Soluciones Avanzadas. Julio de 1994. (1994).

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

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

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

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

[15] NAUR, PETER and BRIAN RANDELL (eds.). 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. (1969).

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

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

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

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

[20] GÓMEZ-PÉREZ, A., FERNÁNDEZ-LÓPEZ, M., and CORCHO M. Ontological Engineering. Springer Verlag London. (2004).