INNOVA Research Journal, ISSN 2477-9024
(Octubre, 2018). Vol. 3, No.10 pp. 20-33
DOI: https://doi.org/10.33890/innova.v3.n10.2018.651
URL: http://revistas.uide.edu.ec/index.php/innova/index
Correo: innova@uide.edu.ec

Evolución de las Metodologías y Modelos utilizados en el Desarrollo de Software.

Evolution of the Methodologies and Models used in Software Development.

Johanna Patricia Zumba Gamboa
Universidad de Guayaquil, Ecuador
Cecibel Alexandra León Arreaga
Universidad de Guayaquil, Ecuador

Autor para Correspondencia: johanna.zumbag@ug.edu.ec; cecibel.leona@ug.edu.ec
Fecha de recepción: 24 de julio de 2018 - Fecha de aceptación: 20 septiembre de 2018

Resumen: las metodologías de Desarrollo de Software (DS.) han experimentado un proceso histórico y evolutivo que inicia en los años 40 con la aparición de las primeras computadoras, entonces no se contaban con parámetros ni estándares, el DS. Era prácticamente empírico y artesanal lo que llevó a que una buena parte de los proyectos fallaran en cubrir las exceptivas de los usuarios, así como en entregas extemporáneas y presupuestos excedidos, sobreviniendo la "crisis del Software" la respuesta para superarla fue la adopción de modelos y metodologías clásicas que progresivamente fueron incorporando estándares, controles y formalidades al DS. En un afán que llegó a ser definido como "triángulo de hierro." La evolución no se detuvo, con la llegada del Internet surgen proyectos caracterizados por requerimientos cambiantes y tiempos de entregas breves para los que las metodologías existentes no se adaptaban idóneamente, surgen las metodologías ágiles, enfocadas en interacción equipo-usuarios, entregas tempranas y adaptación a los cambios; conviven con los esquemas tradicionales y agrupan a comunidades activas. Este esfuerzo documental busca reseñar de manera integral todo ese cambio evolutivo, por cuanto la mayoría de los trabajos en el área se enfocan en divulgar los métodos ágiles dejando subestimada su procedencia.
Palabras Clave: Metodologías de Desarrollo de Software; Metodologías Ágiles; Modelos.

Abstract: The Software Development methodologies have experimented a historic and evolutionary process that started in 1940 with the first Computers. At these times there were not standards or parameters. The software development was empiric or artisanal, facts which brought as a consequence a "Software Crisis" because the projects don't cover the client's expectations or were delivered postdates and very expensive. The answer was the boring of models and classics methodologies, these were given discipline and rigor; standards and controls to the process of software development. Maybe in an exaggerated way, at this point were named "The steel triangle." The evolution didn´t stop, the arrival of Internet brought a lot of projects with changing requirements, and very short deadlines, for these the classical Methodologies did not adjust at all. Agile Methodologies emerged, Focused on interactions between clients and team work, earlier delivers and welcome the changes; they coexist with the classics schemes and attract active and growing communities. This paper represents an effort to review on a holistic way all of this evolutionary change, because many papers focus on divulge Agile Methodologies leaving out of its provenance.
Key Words: Software Development methodologies; Agile methodologies; Models.

Introducción

Desde la aparición de los primeros programas computacionales hasta la actualidad, el desarrollo de software ha pasado por un proceso evolutivo tan marcado y propio de cualquier área tecnológica.

Garcés y Egas (2015) sostienen que en éste campo, se ha trascendido desde unos primeros ejercicios de codificación, en la mayoría de los casos costosos y alejados de la expectativa del usuario, pasando por modelos de codificar y corregir, hacia métodos que fueron agregando rigurosidad, enfocados en actividades y procesos (Cadavid, Martínez & Velez. 2013) marcos secuenciales y en una sola dirección que buscaban llevar a término el proyecto bajo restricciones de tiempo y presupuesto.

Se gestaron metodologías las tradicionales, según Avison y Fitzgerald (1995) citados por Tinoco, Rosales y Salas (2010) en respuesta a una "crisis del Software" producto de excesivos costos, entregas extemporáneas y un limitado cubrimiento a las expectativas de los clientes, se formalizan como compendios de criterios, procedimientos, técnicas, herramientas y documentos para guiar a los desarrolladores de software en sus esfuerzos por la implementación de sistemas de información.

El énfasis en la arquitectura, el exceso de controles y la formalidad de la documentación, llevaron a restar aplicación a este tipo de metodologías en contextos cambiantes de una necesaria y rápida adaptación (Canós, Letelier & Penades, 2003) sobrevenidos por la llegada de tecnologías comunicacionales como la Internet, dando paso a criterios alternos en el éxito del desarrollo de software, según Tinoco et al. (2010), lo que marca la llegada de las metodologías ligeras, llamadas luego ágiles en respuesta a la burocracia que marcó a los primeros movimientos metódicos, éstas se concentran en una mayor interacción con el cliente, mayor importancia hacia el individuo y entregas periódicas e incrementales.

Aunque varios son los movimientos que se alinean al manifiesto ágil, como eje central que resume los valores y principios de estas metodologías, la programación extrema o "Extreme Programing" (XP) y Scrum lucen como los movimientos metodológicos de mayor presencia y aplicación. (Díaz 2009; Tinoco, Rosales & Salas, 2010; Pérez, 2011; Cadavid, Fernández & Morales, 2013)

El nivel de aceptación y las experiencias reales de implementación en proyectos relativamente pequeños, masivos y de requerimientos cambiantes, es tal que las metodologías ágiles han logrado agremiación y han consolidado comunidades activas como la Agile Alliance (Tinoco et al. 2010), Scrum Manager y la Agile Spain en el caso Ibérico, según Díaz (2009).

Las metodologías Agiles, más que un marco de procedimientos y herramientas se conciben como una filosofía, una práctica orientada a las personas no a los procesos, (Cadavid, et al. 2013) en prospectiva se mueven hacia un ambiente de "Post-agilismo" integrando a las escuelas ya posicionadas herramientas como el Kanban, el visual thinking y graphic design; así quedó explícito en temarios de eventos recientes como la conferencia Agile Spain 2017 celebrada en noviembre en Sevilla, España.

Este esfuerzo documental representa una iniciativa por reseñar la evolución de las metodologías de desarrollo de software de forma integral, por cuanto una parte importante de los documentos revisados hace énfasis casi exclusivo en las metodologías ágiles más conocidas (XP y Scrum), por lo que se espera brindar un panorama amplio e incluso comparativo, de distintos métodos de desarrollo. En una primera sección se describen los primeros pasos del desarrollo de software y las coyunturas que llevaron a establecer los métodos, en la segunda parte se presentan los enfoques tradicionales, modelos y metodologías más usados, haciendo una diferenciación entre ambos y en una tercera sección se enlistan las metodologías ágiles, haciendo referencia a las más usadas como parte de los movimientos comunitarios e iniciativas más actuales en éste campo de la ingeniería del software.

Los Orígenes (1940-1960)

La década de los 40 marca el inicio de la primera generación de computadoras y con ellas la serie de programas y sistemas que éstas requieren para funcionar, (Rivas, Corona, Gutiérrez & Hernández, 2015). Según Garcés & Egas (2015), las primeras prácticas de desarrollo no obedecían a una metodología, los llamados programadores se abocaban a desarrollar sus códigos una vez que comprendían los requerimientos de sus clientes.

Aún hacia finales de los 50, la producción del software era "totalmente artesanal y en ella no habían metodologías definidas, lo que acarreaba multitud de problemas." Carballar (2009). Estos primeros empirismos se enfocaban en la tarea de codificar, más que comprender, diseñar o documentar los requerimientos de los usuarios.

Para la época, se inicia el empleo masivo del hardware y la complejidad de las tareas que realizan los equipos computacionales existentes va en aumento, imprimiendo esa complejidad a las tareas de programación, se hizo necesario estandarizar y simplificar dichas actividades, lo que da cabida a los leguajes de programación. Una primera generación es llamada lenguaje de máquina, seguido de una segunda denominada lenguaje ensamblador. La tercera generación o lenguajes de programación de alto nivel, simplifica aún más los procesos de codificación puesto que las instrucciones son comprensibles por el programador casi como si fuesen leguajes naturales. (Garcés & Egas, 2015).

Aunque se disponían de importantes técnicas de codificación para la época, clientes y usuarios en una parte importante de los desarrollos no quedaban satisfechos con los sistemas, pues 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. Ahora se empieza a hablar de analistas programadores y analistas de sistemas." (Carballar, 2009); esta diferenciación empieza a marcarse hacia mediados de los años 50.

Aún ante estos iniciales pero significativos avances, los resultados finales eran impredecibles. No se tenía una manera clara de controlar los avances de los proyectos, aunque se estructuraban equipos de trabajo, no existían fases diferenciables ni productos intermedios para poder realizar verificaciones. Los desarrollos tomaban curso sin mediciones, para un director era casi imposible tomar decisiones.

Según Carballar (2009), ante este panorama, la detección tardía de defectos era casi inevitable, en la mayoría de los casos cuando se estaba probando el código o aún peor, cuando el sistema se encontraba en producción, por lo que los desarrollos resultaban costosos.

Finales de los 60, marca el inicio de una concienciación sobre la necesidad de establecer controles o puntos de verificación del correcto avance de los sistemas, evitando errores tardíos, así como la necesidad de documentación y mayores estándares.

Modelos y metodologías Clásicas, (desde 1960.)

El ambiente caótico que caracterizó las primeras etapas de las técnicas de codificación desembocó en la inconformidad creciente de los usuarios, así como en proyectos excedidos en tiempos de entrega y presupuestos, casi de forma masiva, causas que marcaron la pauta en la "búsqueda de alternativas para esquematizar de alguna manera la producción del Software" (Garcés & Egas, 2015).

El inicio de los 60 acoge a un primer modelo conocido como "Code and Fix" o codificar y corregir, emerge como un modelo que representa más de la anarquía de los primeros años, sin embargo, personifica el afán de muchos equipos de programación por adoptar una serie de pasos formales a seguir. Según Garcés y Egas (2015) se consideró como "una base inicial para la fabricación del Software."

Sentó las bases para el establecimiento de la idea general de los requerimientos, el diseño, codificación, la depuración y los métodos de prueba todo en un solo sentido, hasta obtener el entregable, si al final del proceso se obtenía un diseño incorrecto, se desechaba y el proceso volvía a su fase inicial. "el modelo implementaba el código y luego se pensaba en los requisitos, diseño, validación y mantenimiento." (Garcés & Egas, 2015).

Mediados de los 60 toma popularidad el término "crisis del Software" para referirse a los problemas recurrentes que caracterizaba los procesos de desarrollo, en costos, fiabilidad, entregas extemporáneas e insatisfacción de los clientes; aspectos que se acuñaron como "síntomas de la crisis del Software." (ob. Cit.) Entre otras dificultades prácticas que acusaban esta la alta calificación que demandaban los sistemas para con los usuarios y lo costoso y complejo que resultaba su mantenimiento.

Algunos de estos proyectos eran tan críticos, como, por ejemplo, sistemas de control de aeropuertos, equipos para medicina; que sus implicaciones iban más allá de las pérdidas millonarias que causaban. (Patponto, 2010).

En 1968, tras una conferencia en Garmisch (Alemania) surge el término "Ingeniería del Software" como un "enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software" (Patpondo, 2010) y en respuesta a la crisis del software.

La industria del Hardware seguía creciendo, con procesos estandarizados, divididos en tareas u operaciones, documentos, manuales y certificaciones de calidad; el software parecía no alienarse a esa rigurosidad, en puertas de los años 70, los métodos de desarrollo aún eran informales. En 1972 Edsger Dijkstra, presenta su trabajo titulado "The Humble Programmer" y

sienta las bases para la creación de las metodologías tradicionales conocidas y aún usadas hasta hoy. (Tinoco et al. 2010)

Dijkstra junto a otros autores publicaría luego el artículo "Go to statement considered harmful" y junto a su libro "Discipline of Programming" estableció ciertos parámetros para el desarrollo del Software de forma exitosa que actualmente siguen siendo legado, algunos de sus postulados resumidos en Tinoco el al. (2010):

- *El coste del desarrollo inicial debe ser relativamente bajo.*
- *El software debe ser fácil de mantener.*
- *Cualquier desarrollo debe de ser portable a nuevo hardware.*
- *El software debe hacer lo que el cliente quiere.*
- *El desarrollo de programas mediante top-down y en contraposición a bottom-up.*
- *El desarrollo debe seguir un conjunto de pasos formales para descomponer los problemas grandes (lema divide y vencerás).*

El término ingeniería del software empezó a usarse con más énfasis a finales de la década de los 70, para representar ya para ese momento un área de conocimiento que se estaba desarrollando en torno a las problemáticas que ofrecía el software y tomando base en los postulados de Dijkstra. (Cendejas, 2014).

La búsqueda progresiva de solución a los problemas del Software de la década de los 70, dio importancia al análisis de los códigos, el uso de los postulados de Dijkstra permitió solucionar inconvenientes de sistemas complejos a través del análisis por partes o etapas, se introducen hacia la segunda mitad de esta década la planificación y la administración.

Garcés & Egas (2015). Reconocen la aparición en 1975 del "Software Develop Life Clicle" (SDLC) o clico de vida del desarrollo del Software, como un consenso formal para la construcción de sistemas, sienta las bases de los estados por los que debe trascender un producto de desarrollo desde que nace a partir de un requerimiento, hasta que muere luego de su mantenimiento. Las fases o estados se muestran en la figura 1.

Figura 1: Ciclo de desarrollo del software.
Fuente: http://upsg01.foroactivo.com

Se concibe análogamente el concepto de "proceso de desarrollo del Software" en procura de transformar una necesidad en una solución automatizada de manera eficiente (ob cit.)0

Para finales de la década de los 70, la ingeniería del Software se refuerza mediante la implementación de una serie de "modelos" que dividen el proyecto en etapas desde su concepción inicial, el desarrollo, pruebas, lanzamiento y mantenimiento. Para cada etapa se crean normativas y parámetros procurando orientar los ajustes a la necesidad de los clientes y un cambio general en aquella visión de la crisis del software, se convierten en guía para los ingenieros orientando el quehacer de las distintas actividades técnicas y suministrando un marco formal para la administración, avances en el desarrollo y mantenimiento; estimación de recursos y definición de puntos de control.

Aparecen entre 1970 y 1988 los "modelos tradicionales de desarrollo de software." En este punto es importante hacer una diferenciación entre "modelos" y "metodologías", términos que han sido usados incluso de manera indistinta por algunos autores, Cendejas (2014) en su tesis doctoral deja clara la separación de ambos conceptos argumentando que:

> **"Modelo de desarrollo de software:** es una representación simplificada del proceso para el desarrollo de software, presentada desde una perspectiva específica. Mientras que la **Metodología de desarrollo de software:** es un enfoque estructurado para el desarrollo de software que incluye modelos de sistemas, notaciones, reglas, sugerencias de diseño y guías de procesos." (p.87)

El autor aclara que modelo es la abstracción simplificada del proceso de desarrollo concebido desde una perspectiva particular (Cascada, desarrollo evolutivo, componentes) mientras que la metodología es un compendio amplio que incluye al modelo, reglas, notaciones y documentación del proceso.

Otra definición de "modelo de DS" oportuna y que circunscribe el alcance del término, es ofrecida por Cervantes & Gómez (2012) argumentando que "el modelo es una representación abstracta del proceso determinativa del orden en que se llevan a cabo las actividades del proceso de DS.

En el cuadro 1 se recopilan los principales modelos y metodologías tradicionales concebidos entre 1970 y 1990, en un orden no exhaustivo.

Modelos de DS.	Metodologías Tradicionales de DS.
• Modelo de Cascada • Modelo Cascada en "V" • Modelo de Desarrollo Evolutivo (Espiral) • Modelo de Desarrollo Evolutivo por Prototipos • Desarrollo Evolutivo por etapas o Incremental • Desarrollo Evolutivo Iterativo • Modelo Basado en Componentes	• RUP (Rational Unified Process) • RAD (Rapid Application Development) • MSF (Microsoft Solution Framework) • Win-Win Spiral Model • Iconix • Desarrollo de sistemas de Jackson (JSD). • Ingeniería de la información. • Structured System Analysis and Design Method (SSADM).

Cuadro 1: Modelos y Metodologías Tradicionales de Desarrollo de Software (DS).
Fuente: Cervantes & Gómez (2012); Cendejas (2014); Garcés & Egas (2015); Patpondo (2010); Rivas et al. (2015) y recopilación propia.

Por razones de extensión del documento, solo se ofrecerá una breve caracterización de los modelos y de las metodologías más usadas.

Modelo en Cascada. Propiciado por Winston Royce en 1970, sugiere un enfoque sistemático y secuencial, disciplinado y basado en análisis, diseño, pruebas y mantenimiento. Al final de cada etapa se reúnen y revisan los documentos para garantizar que se cumplen los requerimientos antes de avanzar a la fase siguiente (Garcés & Egas, 2015). Pionero en guiar el proceso de DS dirigido por un plan, introduciendo una planificación de casa fase antes de empezar a trabajar en ella.

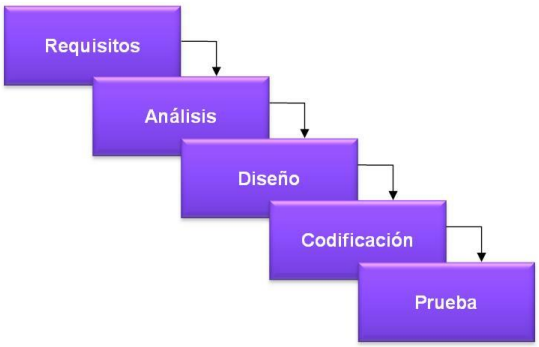

Figura 2: Esquema del modelo en Cascada.
Fuente: www.iedge.eu

Modelo de Cascada en "V" propuesto por Alan Davis a principios de los 90. Se base en el modelo en cascada con la innovación de procurar actividades de pruebas más efectivas y productivas mediante la introducción de validaciones en la medida en que se avanza en el proyecto; dado que en el modelo tradicional las pruebas se introducían al final los defectos aparecían en forma tardía. Las pruebas necesitan empezarse lo más pronto posible en el ciclo de vida y estas actividades deberían ser llevadas a cabo en paralelo con las actividades de desarrollo. (Sáez, Rodríguez, Villanueva & Cueto, 2014).

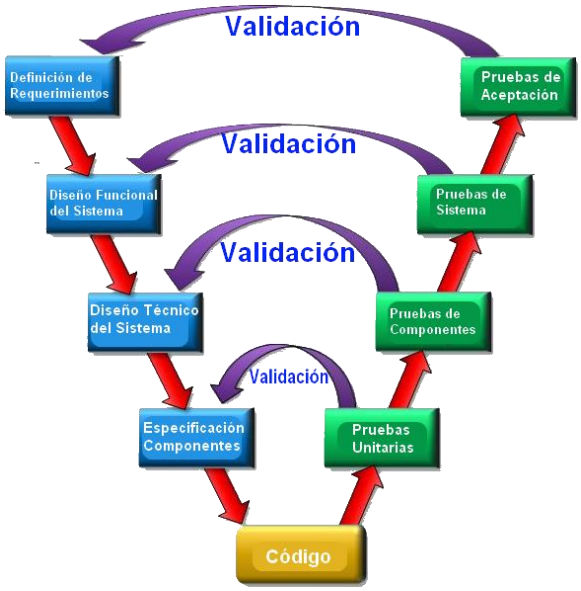

Figura 3: Esquema del modelo en "V"
Fuente: https://ingsoftware.weebly.com

Modelo de Desarrollo Incremental. Harlan Mills en el año 1980. Se basa en el desarrollo a partir del incremento de la funcionabilidad del programa, se puede considerar un precursor de las modernas metodologías iterativas. El primer incremento es a menudo un desarrollo esencial, apenas con los requisitos básicos, cada incremento representa una entrega escalable. Los primeros incrementos son versiones incompletas del producto final, pero proporcionan al usuario la funcionalidad. (Sáenz et al, 2014). El esquema de este sencillo, pero procedimental modelo se muestra en la figura 4.

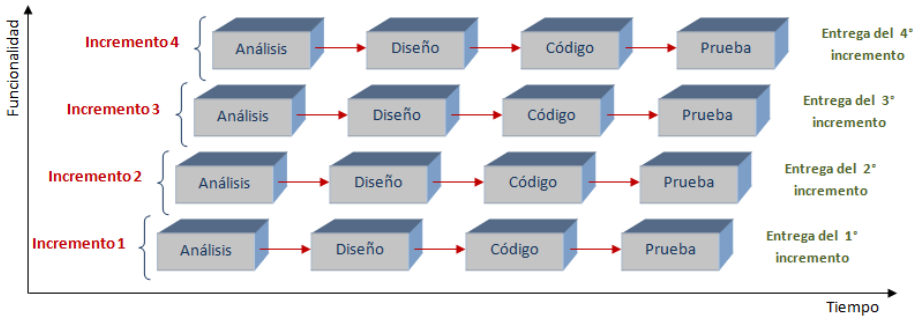

Figura 4: Esquema del modelo Incremental.
Fuente: https://procesosoftware.wikispaces.com

Modelo de desarrollo evolutivo (espiral). Propuesto por Barry Boehm en 1986, en su artículo "A Spiral Model of Software Development and Enhancement" (Patpondo, 2010). Conjuga una naturaleza iterativa en la construcción de prototipos con aspectos controlados y sistemáticos del modelo en cascada. (Cendejas, 2014.) Cuando se aplica este modelo en espiral, el software se desarrolla en una serie de entregas evolutivas. Cada una de las actividades del marco de trabajo representan un segmento de la ruta del espiral. En cada ciclo repetitivo va ganando madurez el producto final.

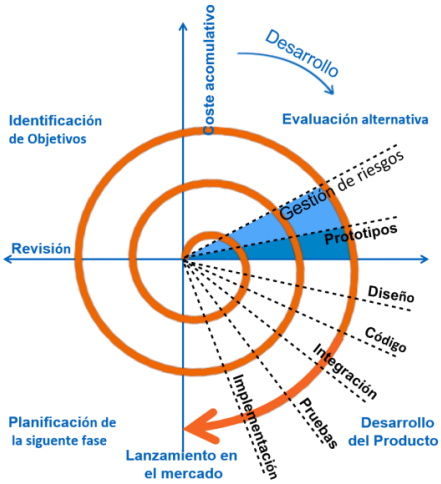

Figura 5: Esquema del modelo en Espiral
Fuente: http://www.tutorialspoint.com

Modelo evolutivo por prototipos. Modelo iterativo a través del cual es posible identificar los requerimientos del cliente mediante la construcción de un prototipo de funcionalidad simulada; si no se ajusta a la expectativa del cliente, se construye otro prototipo con una definición mejorada, el diseño va evolucionando ajustándose cada vez más al requerimiento, aunque su funcionalidad será simulada hasta tanto se aclaran la totalidad de los requerimientos con la validación del último prototipo. (Cervantes & Gómez, 2012.)

Modelo de desarrollo basado en Componentes. Divulgada por Fred Brooks en 1987, originalmente bajo la filosofía, "compre y no construya." Promueve el uso de desarrollos "pre-fabricados" que se pueden reutilizar, con ello se emplean arquitecturas, diseños y software de una aplicación para construir otra. Representa un paso importante hacia la agilización. El famoso autor Sommerville (2005) en su libro de texto ingeniería del software argumenta que en la mayoría de los proyectos existe reutilización de software cada vez que los desarrolladores y otros involucrados en el proyecto conocen los diseños y su similitud a los requerimientos, los buscan, los modifican y los incorporan en el sistema.

Hasta ahora se han reseñado los modelos de desarrollo de software, como abstracciones del proceso, ahora bien las metodologías representan un paso más ya que no solo se centran en esquematizar las etapas del proceso, sino que reúnen la manera de gestionar la documentación, los procedimientos, técnicas y herramientas auxiliares etapa por etapa.

Sommerville (2005) resume el término "Metodología" en desarrollo de software como un "enfoque estructurado" cuyo propósito es facilitar la producción de un software de calidad a un costo razonable. Implica: procesos, modelos, tareas, procedimientos y herramientas.

En cuanto a metodologías hay que hacer una distinción entre las tradicionales, formales o "primeros enfoques estructurados" y las "ágiles" que surgen en contraposición a las primeras. Los enfoques tradicionales se gestan desde los años 80, de forma paralela con muchos de los modelos ya expuestos.

Según Pérez (2011) los marcos tradicionales imponen una planificación rígida y meticulosa del proyecto, soportada por herramientas y una carga de trabajo pesada en planificación, diseño y documentación, en un afán por hacer al desarrollo predecible dentro de un marco de temporalidad y costo; recordando que derivan de solventar los problemas de la crisis del software vivida décadas atrás. Las más difundidas y citadas en textos son: Rapid Application Development (RAD), Rational Unified Process (RUP) y Microsoft Solution Framework (MSF).

Rapid Application Development (RAD). O desarrollo rápido de aplicaciones, presentado por James Martin desde 1980, formalmente documentada en su libro con ese mismo nombre en 1991. La metodología se centra en una lista de tareas y una estructura de desglose del trabajo orientada a la rapidez, aunque no está alineada al "manifiesto ágil" si buscó responder a la necesidad de agilizar las entregas de aplicaciones. Comprende el desarrollo bajo un modelo iterativo, la construcción de prototipos y el uso de utilidades CASE (Computer Aided Software Engineering) esto es, aplicaciones informáticas dirigidas a aumentar la productividad en todos los aspectos del ciclo de desarrollo (Garcés & Egas, 2015).

Rational Unified Process (RUP). Propuesta en 1998 por Ivar Jacobson, Grady Booch y James Rumbaugh. (Pérez, 2011). Metodología basada en los modelos en Cascada y por Componentes. Presenta las siguientes características: Es dirigido por "casos de uso" esto es, la descripción del servicio que el usuario requiere del sistema y la secuencia de iteraciones usuario-sistema; se centra en la arquitectura, dicta pautas específicas para la constitución del equipo y las escalas de tiempo, es iterativa e incremental.

Es una de las metodologías clásicas vigentes y más usadas para el análisis, desarrollo y documentación de sistemas orientados a objetos, muy aplicada en proyectos de gran complejidad y magnitud con apoyo de equipos expertos, desde sus inicios ha contado con respaldo por parte de IBM. (ob. Cit.)

Microsoft Solution Framework (MSF). Pérez (2011) señala que fue introducida por primera vez en 1994 como un conjunto de las mejores prácticas (principios, modelos, disciplinas, conceptos y directrices) en los desarrollos de Software de Microsoft y Microsoft Consulting Service. Es flexible, permite aplicar de manera individual e independiente cada uno de sus componentes, es escalable según la magnitud del proyecto; fundamentada en los modelos espiral y cascada. Profesa la aplicación de 8 principios fundamentales para una mejor organización del trabajo; modelos o esquemas para la organización de los equipos y disciplinas de gestión. (ob. Cit.)

Metodologías Ágiles, desde 1990

Los años 90 marca la masificación del Internet y con ello el florecimiento de un sinfín de requerimientos rápidos e imprecisos, lo que demanda celeridad al desarrollo de software.

Al implementar las metodologías clásicas en proyectos medianos con mayores exigencias en tiempos de respuesta y requerimientos imprecisos y cambiantes, se obtuvieron resultados ineficientes, debido a que se pasaba más tiempo pensando en el diseño y los controles que en hacer frente a posibles cambios en las especificaciones; estos no eran compatibles con la manera de realizar los análisis y la documentación, haciendo del desarrollo de software un proceso improductivo e ineficiente. (Pérez, 2011).

Aparecen en respuesta las metodologías ágiles para el desarrollo de software, alternativas que procuran un enfoque en el software y no en la arquitectura o la documentación, con un enfoque iterativo dan la bienvenida a los requerimientos cambiantes y entregas funcionales desde etapas tempranas con la participación del cliente.

Una buena diferenciación entre las metodologías tradicionales y las llamadas ágiles se presenta en el cuadro 2, originalmente recabada por Canós et al. (2003) y ampliada por Cendejas (2014).

Metodologías ágiles	Metodologías tradicionales
Se basan en heurísticas provenientes de prácticas de producción de código	Se basan en normas provenientes de estándares seguidos por el entorno de desarrollo
Énfasis en los aspectos humanos: el individuo y el trabajo en equipo.	Énfasis en la definición del proceso: roles, actividades y artefactos.
Preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente por el equipo	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso muy controlado, numerosas normas
Contrato flexible e incluso inexistente	Contrato prefijado
El cliente es parte del desarrollo	Cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10) con pocos roles, más genéricos y flexibles.	Grupos grandes con Más roles y más específicos.
Orientada a proyectos pequeños, y en el mismo lugar.	Aplicables a proyectos de cualquier tamaño, pero suelen ser especialmente efectivas/usadas en proyectos grandes.
Pocos artefactos El modelo es prescindible, modelos desechables.	Más artefactos. El modelo es esencial, mantenimiento en los modelos.
Menor énfasis en la arquitectura del software, se va definiendo y mejorando a lo largo del proyecto.	La arquitectura del software es esencial, se defina tempranamente en el proyecto.

Cuadro 2. Comparación de metodologías ágiles versus tradicionales.
Fuente: Canós *et al. (*2005); Cendejas (2014) y elaboración propia.

Este contraste se puede sintetizar afirmando que los enfoques ágiles valoran a los individuos, equipo e interacciones por encima de los procesos y herramientas, al software en funcionamiento por encima de la documentación excesiva, a la colaboración con el cliente por encima de un contrato suscrito y la adaptabilidad en vez que un plan riguroso. (Cadavid et al. 2013). Tal enumeración corresponde a los cuatro principios básicos del manifiesto ágil.

Amaya (2013), Tinoco et al. (2010) y Rivas et al. (2011) conjugan una completa recopilación de los métodos que se alinean a la filosofía ágil, estos se listan en orden alfabético en el cuadro 3:

1. Adaptative Software Development	11. Internet Speed Development
2. Agile Modeling	12. Lean development
3. Agile Model Driven Development	13. Mobile-D
4. Agile Project Management	14. Open Unified Process
5. Agile Unified Process	15. Pragmatic programming
6. Crystal Methods	16. Scrum
7. Dynamic Systems development methods	17. Story cards driven development
8. Evolutionary Project Management	18. Test Driven Development

9. Extreme Programming	19. Win Win Spiral
10. Feature Driven Development	20. X-Breed

Cuadro 3. Enumeración de las Metodologías ágiles
Fuente: Amaya (2013); Tinoco et al. (2010); Rivas et al. (2011), recopilación propia.

Todas las metodologías listadas arriba tienen como común denominador su alineación al llamado "manifiesto Ágil", una declaración formal firmada en febrero del 2001 en Utah, USA, por 17 representantes de la industria del software, (Canós et al. 2003). Reunión en la que reafirman los valores y principios que se contrastan en el cuadro 2. Desde entonces crean la "Agile Alliance" (https://www.agilealliance.org; @agilealliance) una de las más activas congregaciones dedicadas a promover la filosofía del desarrollo ágil, actualmente vigente y con representación mundial.

De los veinte movimientos a los que se hace mención en el cuadro 3, son dos las de mayor presencia internacional, mayor divulgación en textos y en la web, certificaciones, training, comunidades activas y participación en eventos: Extreme Programming y Scrum. (Tinoco et al. 2010 y Cadavid et al. 2013).

Extreme Programming. Propuesta por Kent Beck, en su trabajo fundamental que publicó en 1999 buscando guiar a equipos de desarrollo de software pequeños, entre dos y diez desarrolladores, en ambientes de requerimientos imprecisos o cambiantes. (Cadavid et al. 2013). Cinco valores fundamentan sus principios: Simplicidad, Comunicación, Retroalimentación, Respeto y Coraje. Sus postulados o principios son: Retroalimentación rápida, asumir simplicidad, el cambio incremental, la aceptación del cambio y el trabajo de calidad.

Esta serie de valores y principios derivan en "prácticas XP" que son la aplicación de la metodología: "planning game" (la definición del alcance y fechas de cumplimiento); pequeñas entregas (iteraciones); diseño tan simple como sea posible; programación en pareja, pruebas como guías del desarrollo; "refactoring" o cambios que mejoren la estructura del sistema; integración continua; propiedad común del código, ritmo de trabajo a paso sostenible, cliente en sitio, "metáfora" o abstracción conjunta del sistema y estándares o reglas de código.

Scrum. Su nombre no es una sigla, sino un término deportivo aplicable al rugby. Su primera referencia en el contexto de desarrollo data de 1986; utiliza un enfoque incremental que tiene como fundamento la teoría de control empírico de procesos. Los llamados Equipos Scrum son autogestionados, multifuncionales y trabajan en iteraciones.

Define tres roles: el "Scrum master", líder del equipo y de la implementación de la filosofía, mas no del desarrollo; el dueño del producto y el equipo de desarrollo. El "dueño del producto" que representa a los interesados y vela por la maximización del valor del entregable y "el equipo de desarrollo" responsables de convertir los requerimientos del cliente en iteraciones funcionales del producto. (Cadavid et al. 2013).

La terminología Scrum define un evento temporal conocido como "Sprint" con una duración máxima de un mes en el que debe crearse una versión utilizable del producto. Cada Sprint cumple con los siguientes elementos: reunión de planeación, Daily Scrum, trabajo de desarrollo, revisión del Sprint y retrospectiva del Sprint.

La terminología también involucra "Artefactos de Scrum." Estos son subproductos de las actividades de la metodología que le brindan dirección y transparencia al equipo son: Product Backlog, Sprint Backlog, Monitoreo de Progreso e Incremento. (ob. Cit.) Toda esta terminología se esquematiza en la figura 6.

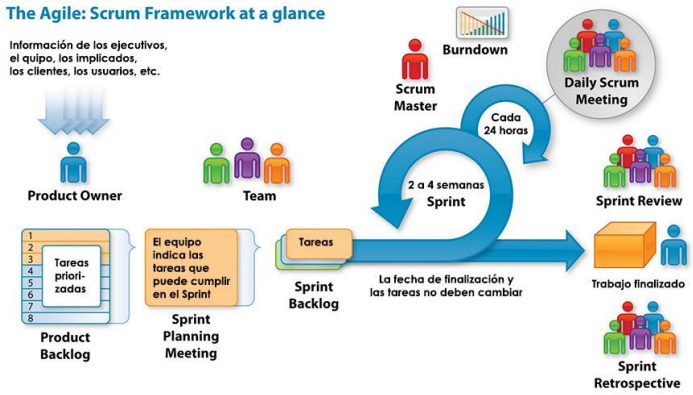

Figura 6: Esquema de la Metodología Scrum.
Fuente: http://metodologiascrum.readthedocs.io

Las metodologías ágiles no son la solución definitiva a los problemas de Desarrollo, ni tan siquiera se pueden aplicar en todos los casos; pero sí aportan otro punto de vista de cómo se puede desarrollar sistemas, de forma más rápida, más adaptable y sin tener que perder la rigurosidad de que imponen las metodologías clásicas. (Gonzales, 2013).

Conclusiones

El principal propósito de esta entrega ha sido reseñar la evolución del desarrollo del software desde sus inicios en la década de los 40 hasta los actuales momentos. Un proceso caracterizado en sus inicios por el empirismo y la improvisación que derivó en un par de décadas en una crisis por la visión generalizada de un software poco ajustado a los requerimientos, costoso y con entregas tardías.

El afán por desvirtuar los síntomas de la llamada "crisis del software" llevo la adopción progresiva de modelos y guías de trabajo cada vez más rigurosas y disciplinadas, centradas en controles exhaustivos de procesos, documentos y roles, prestando limitado interés a los talentos que conforman los equipos. Un interés exacerbado por controlar y hacer predecibles todos los aspectos del desarrollo del software llevó a enmarcar éste proceso en un "marco de hierro" (Díaz, 2009) dominado por fijar un alcance, desarrollando un producto a un costo y tiempo definidos en cumplimiento de un contrato.

Entre los años 80 y 90 metodologías con una carga pesada de trabajo en planificación, diseño y documentación, absorbían gran parte del tiempo destinado al desarrollo del sistema, dejando pocos recursos a la adopción de cambios y al desarrollo en sí. El advenimiento del Internet y de infinidad de proyectos con tiempos de entrega muy limitados y requerimientos cambiantes, genera el caldo de cultivo para el florecimiento de nuevos marcos de acción para desarrollar software, esta vez mas concentradas en los grupos de trabajo y la participación activa

del cliente, apostando a la simplicidad y entregas tempranas más que en diseños, controles y documentos excesivos; aunque esto no representó la pérdida de vigencia de las metodologías formales.

El movimiento ágil sigue cobrando participación entre la comunidad de ingeniería del software, integrando a cada movimiento enmarcado en el manifiesto ágil nuevas adaptaciones de herramientas tan atrayentes como el aprendizaje gráfico, el kanban, las técnicas go to market, en un mosaico que apunta a la flexibilización y la adaptabilidad sin dejar de lado la calidad de la producción.

Bibliografía

Amaya, Y. (2013). Metodologías ágiles en el desarrollo de aplicaciones para dispositivos móviles. Estado actual. *Revista Tecnología.* Vol. 12, No. 2, 111-124.

Cadavid, A., Fernández, D. & Morales, J. (2013). Revisión de metodologías ágiles para el desarrollo de software. *Revista Prospectiva.* Vol. 11, No. 2, 30-39.

Carballar, D. (2009). *Ingeniería del Software.* [Archivo PDF]. Recuperado de http://www.eduinnova.es/dic09/Ingenieria_Software.pdf

Canós, J., Letelier, P. & Penadés, M. (2003). *Metodologías Ágiles en el Desarrollo de Software*. Paper presentado en las VIII Jornadas de Ingeniería del Software y Bases de Datos. Alicante, España. Recuperado de: https://ldc.usb.ve/~abianc/materias/ci4713/actasMetAgiles.pdf.

Cendejas, J. (2014). *Implementación del modelo integral colaborativo (MDSIC) como fuente de innovación para el desarrollo ágil de software en las empresas de la zona centro - occidente en México.* (Tesis doctoral). Universidad Popular Autónoma del Estado de Puebla, México.

Cervantes, J. & Gómez M. (2012). Taxonomía de los modelos y metodologías de desarrollo de software más utilizados. *Revista UDUAL Unión de Universidades de América Latina y el Caribe.* No. 52, 37-47.

Díaz, J. R. (2009). Las metodologías ágiles como garantía de calidad del software. *Revista Española de Innovación, Calidad e Ingeniería del Software.* Vol.5, No. 3.

Garcés, L. & Egas, L. (2015). Evolución de las Metodologías de Desarrollo de la Ingeniería de Software en el Proceso de la Ingeniería de Sistemas. *Revista Científica y Tecnológica UPSE.* Vol. 1, No. 3.

González, J. (2013). *Introducción a las metodologías ágiles. Otras formas de analizar y desarrollar*. [Archivo PDF]. Recuperado de http://openaccess.uoc.edu/webapps/o2/bitstream/10609/63466/2/Técnicas_avanzadas_de_ingeniería_de_software_Módulo3_Introducción_a_las_metodologías_ágiles.pdf

Patponto. (Diciembre, 2010). Ingeniería del Software. [Blog]. Blog Historia dela Informática. Recuperado de https://histinf.blogs.upv.es/2010/12/28/ingenieria-del-software

Pérez, O. (2011). Cuatro enfoques metodológicos para el desarrollo de Software RUP – MSF – XP – SCRUM. *Revista Inventum.* No. 10, 64-78.

Rivas, C., Corona, V., Gutiérrez, J., & Hernández, L. (2015). Metodologías actuales de desarrollo de software. *Revista de Tecnología e Innovación.* Vol. 2, No. 5, 980-986.

Sáez, P., Rodríguez, V., Villanueva, J. & Cueto, M. (2014). *Selección de Modelos y Metodologías Agiles en Proyectos de Software*. Paper presentado en el 18th International Congress on Project Management and Engineering. Alcañiz, España. Recuperado de: http://www.aeipro.com/files/congresos/2014alcaniz/CIDIP2014_1862_1873.4302.pdf

Sommerville, Ian (2005*). Ingeniería del Software*, séptima edición, Madrid: Pearson Addison Wesley.

Tinoco, O., Rosales, P., & Salas, J. (2010). Criterios de selección de metodologías de desarrollo de software. *Industrial Data Revista de Investigación*. Vol. 13, No. 2, 70-74.