<http://www.milenio.com/opinion/varios-autores/universidad-tecnologica-del-valle-del-mezquital/las-metodologias-tradicionales-y-el-desarrollo-agil-de-software>

https://prezi.com/xpo8pdv2ghqm/metodologias-tradicionales-vs-metodologias-agiles/

<http://masteringenieriasoft.blogspot.com/2012/04/metodologias-de-desarrollo-de-software.html>

<https://pmqlinkedin.wordpress.com/about/metodologia-tradicional-o-agil/>

<https://www.quonext.com/blog/metodologias-agiles-scrum-kanban-xp/>

<http://modelo-cascada.blogspot.com/>

Fases del ciclo de vida son ;

Concepción o análisis

Elaboración o diseño

Construcción o programación

Transición o implementación

metodos agiles xp scrum iconix crystal method,aup

retrasar las desiciones

planificación adaptiva permitiendo potenciar aun mas el desarrollo de software a gran escala

permitir la sasticfaccion del cliente

reduce el costo de cambio , reduce los cambios necesarios del proyecto, reduce la toma de decisiones

permite estar preparado siempre para el cambio o las correcciones, y esta siempre planificado el proyecto a corto plazo

la Ingeniería de Software, es una disciplina que ha estudiado propuestas diferentes que permiten apoyar el proceso de desarrollo y construcción de software desde hace muchos años. Sin embargo, algunos de los proyectos de desarrollo fracasan o el resultado final no es el esperado, para el cliente o usuario final, inclusive para los propios desarrolladores, debido a la complejidad inherente del programa que impacta en la calidad y retraso del tiempo esperado o peor aún; porque el software entregado no tiene la funcionalidad requerida que apoye en la solución de los problemas de la empresa que lo solicitó. En los primeros días de la informática, se pensó que el principal motivo por el que los proyectos sufrían retraso y costes excesivos, era el hecho de que la programación no se llevaba a cabo adecuadamente. La solución a esto, fue la introducción de la Programación Estructurada. Este fue sin duda un salto importante, pero pronto se comprobó que los proyectos seguían experimentando retrasos y altos costos. Sin embargo, en la actualidad se han encontrado varios factores que pueden conllevar al fracaso de un proyecto de desarrollo de software. Actualmente, se distinguen 2 grandes grupos: las metodologías tradicionales y las ágiles.

Generalmente, estas distintas metodologías se han basado en un conjunto de actividades básicamente técnicas, perfectamente diferenciadas y relacionadas con: análisis, diseño, programación y prueba. Lo que diferencia a las distintas metodologías es el grado de énfasis que ponen en las distintas actividades. Durante mucho tiempo, se ha utilizado el tradicional modelo de cascada, el cual, ha demostrado que no refleja adecuadamente, la naturaleza compleja del proceso de desarrollo de software. Los problemas que presenta éste modelo nacen de su propia estructura, al ser una secuencia de grandes etapas que requieren como hitos la documentación completa antes de continuar con la siguiente etapa. Esta metodología, es muy práctica para proyectos pequeños y con muy bajos niveles de riesgo, tales como proyectos de nuevas versiones de algún software existente donde haya requerimientos claros y cuya arquitectura e infraestructura de software y hardware no van a cambiar mucho respeto a la versión anterior. Pero una metodología tan inflexible a cambios, como lo fue el modelo tradicional, rápidamente resultó ser impráctica para el desarrollo de nuevos programas de cómputo, que demandan una alta complejidad en su estructura y funcionalidad. Para solucionar este problema, se definieron posteriormente los métodos iterativos o incrementales, que unidos a otras prácticas claves, como la orientación al manejo de riesgos y la planeación adaptable, permiten de forma natural guiar adecuadamente el proceso de desarrollo de software. Actualmente se ha adoptado como estándar en algunas empresas de software la estructura metodológica del RUP de IBM, que organiza el desarrollo de un proyecto informático en iteraciones guiadas en etapas y fases para obtener una arquitectura de software más sólida, robusta, clara y ajustable a las necesidades particulares de cada organización. El proceso Unificado, es un proceso de software genérico que puede ser utilizado para una gran cantidad de tipos de sistemas de software, para diferentes áreas de competencia y diferentes tamaños de proyectos en los que el factor de riesgo tecnológico es muy alto, y los requerimientos son inestables. Provee un enfoque disciplinado en la asignación de tareas y responsabilidades dentro de una organización de desarrollo. Su meta es asegurar la producción de software de muy alta calidad de los usuarios finales dentro de un calendario y presupuesto predecible. Sin embargo, el proceso unificado no ha sido aplicado en muchas empresas por su complejidad de administración y el tiempo que requiere ser invertido en la documentación, desaprovechado sus considerables ventajas. Es así, como aparecen los defensores del desarrollo ágil del software, que resaltan la importancia de los talentos y habilidades de los individuos, más que las metodologías y documentación. Las metodologías ágiles que se centran especialmente en el factor humano o el producto de software, dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con interacciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo manteniendo una alta calidad. Las metodologías ágiles, están revolucionando la manera de producir software, y a la vez generan un amplio debate entre sus seguidores, quienes por escepticismo apoyan las metodologías tradicionales. Su objetivo es esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto, pretendiendo ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas.

**METODOLOGIAS DE DESARROLLO DE SOFTWARE TRADICIONALES VS AGILES**

**1.**                  **Resumen.-**

Desarrollar software implica muchas cosas, desde su planificación hasta la puesta en marcha se deben de seguir un sinnúmero de pasos o actividades. Hoy en día existen diversas metodologías para hacerlo, sin embargo es necesario definir primero la naturaleza del software antes de elegir un determinado ciclo de vida.

En el presente trabajo se detallan los dos grandes enfoques, tanto metodologías tradicionales y metodologías ágiles, las primeras están pensadas para el uso exhaustivo de documentación durante todo el ciclo del proyecto mientras que las segundas ponen vital importancia en la capacidad de respuesta a los cambios, la confianza en las habilidades del equipo y al mantener una buena relación con el cliente. Se verán diferencias, ventajas, desventajas y cual puede encajar en un proyecto de software para interés del lector.

***Palabras Claves**** Metodología, RUP, MSF AUP, Scrum, Metodología Tradicional, Metodología Ágil*

**2.**                  **Metodologías tradicionales.-**

Al inicio el desarrollo de software era artesanal en su totalidad, la fuerte necesidad de mejorar el proceso y llevar los proyectos a la meta deseada, tuvieron que importarse la concepción y fundamentos de metodologías  existentes en otras áreas y adaptarlas al desarrollo de software. Esta nueva etapa de adaptación contenía el desarrollo dividido en etapas de manera secuencial que de algo mejoraba la necesidad latente en el campo del software.

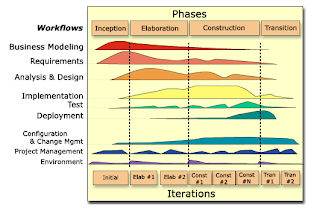
Entre las principales metodologías tradicionales tenemos los ya tan conocidos RUP y MSF entre otros, que centran su atención en llevar una documentación exhaustiva de todo el proyecto y centran su atención en cumplir con un plan de proyecto, definido todo esto, en la fase inicial del desarrollo del proyecto.

Otra de las características importantes dentro de este enfoque tenemos los altos costos al implementar un cambio y al no ofrecer una buena solución para proyectos donde el  entorno es volátil.

Las metodologías tradicionales (formales) se focalizan en documentación, planificación y procesos. (Plantillas, técnicas de administración, revisiones, etc.), a continuación se detalla RUP uno de los métodos más usados dentro de los métodos tradicionales

**Rational Unified Process (RUP)**

Proceso Unificado Rational

[](http://2.bp.blogspot.com/-o7TZY_UlR5k/T5XN3ycmhlI/AAAAAAAAAFA/L0oVT1sS_r8/s1600/rup.png)

RUP es un proceso formal: Provee un acercamiento disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo.

**Fases**Las cuatro fases del ciclo de vida son:

  Concepción

  Elaboración

  Construcción

  Transición

**Ventajas**

  Evaluación en cada fase que permite cambios de objetivos

  Funciona bien en proyectos de innovación.

  Es sencillo, ya que sigue los pasos intuitivos necesarios a la hora de desarrollar el software.

  Seguimiento detallado en cada una de las fases.

**Desventajas**

  La evaluación de riesgos es compleja

  Excesiva flexibilidad para algunos proyectos

  Estamos poniendo a nuestro cliente en una situación que puede ser muy incómoda para él.

Nuestro cliente deberá ser capaz de describir y entender a un gran nivel de detalle para poder acordar un alcance del proyecto con él.

**Microsoft Solution Framework (MSF)**

MSF es un compendio de las mejores prácticas en cuanto a administración de proyectos se refiere. Más que una metodología rígida de administración de proyectos, MSF es una serie de modelos que puede adaptarse a cualquier proyecto de tecnología de información.

**Todo proyecto es separado en cinco principales fases:**

         Visión y Alcances.

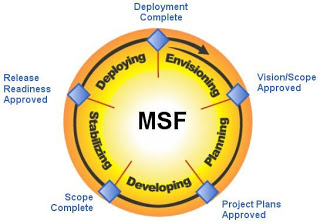
         Planificación.

         Desarrollo.

         Estabilización.

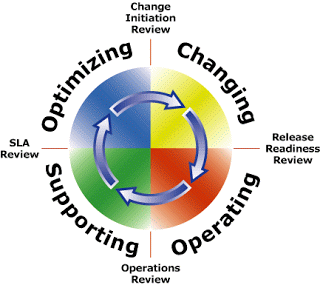
         Implantación.

Modelo de Equipo de MSF

[](http://3.bp.blogspot.com/-G_xaPhA1E4I/T5XN94RQczI/AAAAAAAAAFI/a7LCljO1LTs/s1600/msf.jpg)

**Microsoft Operation Framework.-**

El modelo de proceso MOF está formado por cuadrantes, revisiones de la administración de las operaciones y revisiones de la administración de los servicios. En la figura 1 se muestra el funcionamiento del ciclo de MOF.

[](http://4.bp.blogspot.com/-v0zk7CwHBlo/T5XOFjrdH7I/AAAAAAAAAFQ/W51Y2QIf9uo/s1600/mof.gif)

**Ciclo de Microsoft Operations Framework**

En la figura, se observa que el modelo de proceso MOF se desplaza en sentido de las agujas del reloj y se divide en los cuatro cuadrantes integrados siguientes:

  Cambios

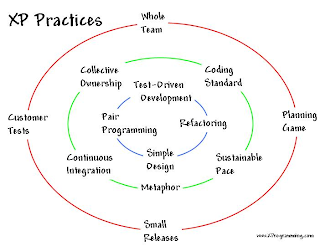
  Operaciones

  Soporte técnico

  Optimización

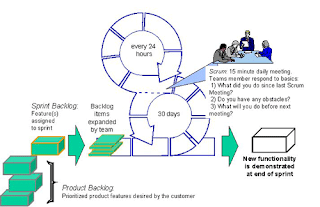
**3.**                  **Metodologías Agiles.-**

**Extreme Programming (XP)**

[](http://3.bp.blogspot.com/-AKlNZ4fznu4/T5XOLmqBhjI/AAAAAAAAAFY/DyAnE2RTt0g/s1600/xp.png)

Los defensores de XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

**Scrum**

[](http://3.bp.blogspot.com/-Kk7PiqzPYoI/T5XOXNtAWlI/AAAAAAAAAFg/ILEBoyRMJM4/s1600/scrum.png)

**4.**      **Diferencias:**

Diferencias entre Metodología Tradicionales y Ágiles

|  |  |
| --- | --- |
| **Metodologías Tradicionales** | **Metodologías Agiles** |
| Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo | Basadas en heurísticas provenientes de prácticas de producción de código |
| Cierta resistencia a los cambios | Especialmente preparados para cambios durante el proyecto |
| Impuestas externamente | Impuestas internamente (por el equipo) |
| Proceso mucho más controlado, con numerosas políticas/normas | Proceso menos controlado, con pocos principios. |
| El cliente interactúa con el equipo de desarrollo mediante reuniones | El cliente es parte del equipo de desarrollo |
| Más artefactos | Pocos artefactos |
| Más roles | Pocos roles |
| Grupos grandes y posiblemente distribuidos | Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio |
| La arquitectura del software es esencial y se  expresa mediante modelos | Menos énfasis en la arquitectura del software |
| Existe un contrato prefijado | No existe contrato tradicional o al menos es bastante flexible |

**5.**      **Conclusión.-**

  El retrasar las decisiones en un proyecto de software permite potenciar el valor del producto tanto para el cliente como al equipo o empresa que desarrolla

  Para que un grupo de desarrollo adopte una metodología ágil debe poseer experiencia  trabajando con metodologías tradicionales, ya que la experiencia es la que predomina en los mementos cruciales del proyecto, además debe tener la capacidad de ser equipos  auto-gestionados, altamente motivados y con gran innovación

  Las metodologías tradicionales son pesadas y que suponen obligatoriamente un “todo o nada”.

  Las metodologías ágiles son más modernas y mejores que cualquiera de las tradicionales.

  Las actividades “de calidad” son inútiles y sólo funcionan en equipos grandes, no se adaptan a nuestros proyectos. Cualquier cosa que nos quite tiempo de tareas técnicas (programar, etc.) es una pérdida de tiempo.

  El uso de metodologías tradicionales es esencial al inicio en un equipo de desarrollo de software

  Las metodologías ágiles se deberían aplicar en proyectos donde exista mucha incertidumbre donde el entorno es volátil, donde los requisitos no se conocen con exactitud, mientras que las metodologías tradicionales obligan al cliente a tomar las decisiones al inicio del proyecto.

### Metodologías tradicionales y metodologías ágiles

Teniendo en cuenta la filosofía de desarrollo de las metodologías, aquellas con mayor énfasis en la planificación y control del proyecto, en especificación precisa de requisitos y modelado, reciben el apelativo de Metodologías Tradicionales o Pesadas. [27]

Estas metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo del software, con el fin de conseguir un software más eficiente. Para ello, se hace énfasis en la planificación total de todo el trabajo a realizar y una vez que está todo detallado, comienza el ciclo de desarrollo del producto software. Se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada [42]. Además, las metodologías tradicionales no se adaptan adecuadamente a los cambios, por lo que no son métodos adecuados cuando se trabaja en un entorno, donde los requisitos no pueden predecirse o bien pueden variar.

Entre las metodologías tradicionales o pesadas podemos citar:

• RUP (Rational Unified Procces)

• MSF (Microsoft Solution Framework)

• Win-Win Spiral Model

• Iconix

Pero sin dudas adaptarse a la agitada sociedad actual implica ser “ágil”, es decir, tener la capacidad de proveer respuestas rápidas y ser adaptables al cambio. Ambas cualidades siempre han sido deseables, pero en el entorno de negocio actual resultan indispensables. Este requerimiento de agilidad en las empresas, gobiernos y cualquier otra organización provoca que el software también deba ser desarrollado de manera ágil.

Las necesidades de un cliente pueden sufrir cambios importantes del momento de contratación de un software al momento de su entrega; y es mucho más importante satisfacer estas últimas que las primeras. Esto requiere procesos de software diferentes que en lugar de rechazar los cambios sean capaces de incorporarlos.

Los procesos ágiles son una buena elección cuando se trabaja con requisitos desconocidos o variables. Si no existen requisitos estables, no existe una gran posibilidad de tener un diseño estable y de seguir un proceso totalmente planificado, que no vaya a variar ni en tiempo ni en dinero. En estas situaciones, un proceso adaptativo será mucho más efectivo que un proceso predictivo. Por otra parte, los procesos de desarrollo adaptativos también facilitan la generación rápida de prototipos y de versiones previos a la entrega final, lo cual agradará al cliente.

Las metodologías ágiles proporcionan una serie de pautas y principios junto a técnicas pragmáticas que puede que no curen todos los males pero harán la entrega del proyecto menos complicada y más satisfactoria tanto para los clientes como para los equipos de entrega. En la figura 1 se muestran los principios que rigen el desarrollo ágil.

Figura 1: Principios del Manifiesto Ágil

Entre las metodologías ágiles más destacadas hasta el momento se pueden nombrar:

• XP (Extreme Programming)

• Scrum

• Crystal Clear

• DSDM (Dynamic Systems Developmemt Method)

• FDD (Feature Driven Development)

• ASD (Adaptive Software Development)

• XBreed

• Extreme Modeling

En el Anexo 1 se muestra una comparación entre las metodologías tradicionales y las ágiles.

## Desarrollo de Software – Metodología Tradicional o Agil ?

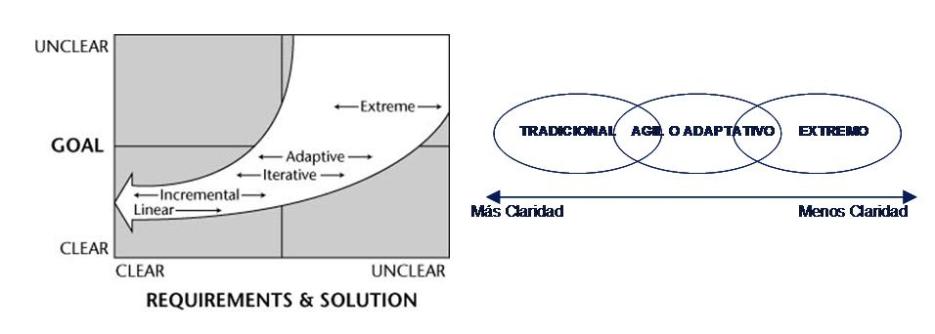
[Deja un comentario](https://pmqlinkedin.wordpress.com/about/metodologia-tradicional-o-agil/#respond)[Go to comments](https://pmqlinkedin.wordpress.com/about/metodologia-tradicional-o-agil/#comments)

***Autor: Norberto Figuerola (PMP)***

Hasta qué punto la metodología tradicional del PMI® es aplicable o no a las actividades de proyectos de desarrollo de software ?.  Esta es una pregunta recurrente y típica que me consultan durante el dictado de distintos cursos y seminarios de Project Manager.En mi experiencia profesional puedo decir que he estado trabajando en la industria de IT durante varios años como Project Manager y he aplicado en mayor o menor medida las prácticas mencionadas en el PMBOK® en casi todos mis proyectos.  De hecho muchas de las compañías en donde trabajé exigen a sus empleados la capacitación formal en la Administración de Proyectos y en algunos casos a obtener la certificación PMP. Tal como lo mencioné en otros artículos el conocimiento metodológico sobre administración de proyectos es un elemento importante que sirve como base para la implementación de una gestión integrada de calidad en cualquier organización, y esto incluye también a las que se dedican a la fabricación de software por supuesto (CMMI es un ejemplo de ello).

De dónde surgen entonces las dudas sobre la aplicación de la metodología PMI® en proyectos de desarrollo de software ?. Pues bien, un hecho que no podemos obviar es que la industria del software es de por si bastante particular y es donde se han producido y seguirán produciéndose cambios muy vertiginosos. De la programación estructurada se ha pasado a la programación orientada a objetos y web, de lenguajes de programación como Cobol se pasó a trabajar con C++ o Java, la gente trabaja y piensa diferente y todo se ve afectado por el slogan de “time to market” lo cual se traduce en entregas lo antes posible a expensas de no ser tan formalistas o documentar.  Estos cambios se traducen en la implementación de las llamadas metodologías ágiles que vienen gestándose y aplicándose desde los años noventa. Con la formación de la Alianza Ágil y subsiguiente publicación del Manifiesto para Desarrollo de Software Ágil se marcó una inflexión en el uso de procesos para desarrollo de software, desatando una fuerte polémica entre promotores y detractores de lo “ágil”.

Como de costumbre se producen errores de interpretación o confusión de conceptos en ambos bandos.  En el libro de Robert K Wysocki  “Effective Project Management” el autor intenta explicar cuando es preferible utilizar una metodología “tradicional” como el PMI o una “ágil” como puede ser Scrum, Crystal, DSDM, o cuando ir a una programación extrema (XP).   Observemos el gráfico que el autor intenta explicarnos:



De acuerdo a su planteo los proyectos los podríamos mapear en el gráfico de más arriba y observaremos que existen proyectos en los cuales los objetivos del negocio, los requerimientos, y la solución a implementarse están mas o menos claros de entrada por lo que se podría planificar todo al principio y sin mayores inconvenientes (lo que Wysocki denomina planificación predictiva o tradicional).  En los otros extremos cuando no existe tanta claridad debería utilizarse una aproximación adaptativa o ágil y cuando hay mucha incertidumbre se recurre a la programación extrema.  Las metodologías ágiles proponen una planificación adaptativa utilizando un ciclo de vida de proyecto iterativo e incremental opuesto al tradicional que utilizaría el ciclo de vida cascada o waterfall.  Por otro lado las metodologías ágiles tratan de bajar los decibeles a todo formalismo que no agrega valor, documentando lo menos posible.

Obviando grandes detalles tales como que el PMBOK® más que una metodología es una guía de mejores prácticas para la gestión de cualquier tipo de proyectos multidisciplinario mundialmente reconocida y utilizada; y que los modelos ágiles más que metodologías son un conjunto de técnicas o herramientas para el desarrollo de software, intentaré explicar mi posición respecto de la pregunta original sobre la utilidad del PMBOK® en proyectos de desarrollo de software y la desmitificación de su tratamiento como metodología “tradicional” no apta para desarrollos ágiles, aclarando que tanto uno como otro son complementarios e importantes de acuerdo al proyecto a encarar.

Para algunos las metodologías ágiles vienen a triunfar donde la metodología “tradicional” había fracasado. Para otros, las metodologías ágiles son sólo una excusa para desarrollar software sin planes, diseño ni documentación. Ambos extremos son erróneos; por una parte, varias prácticas ágiles tienen nichos donde funcionan muy bien, pero hay otros ambientes donde pierden su efectividad o deben ser modificadas para lograr el objetivo. Tampoco es correcto que lo ágil se contraponga con la aplicación de mejores prácticas en gestión de proyectos.

**Los Ciclos de Vida del Proyecto:**Uno de los principales errores es que para muchos el único ciclo de vida conocido es el de cascada, donde se ejecutan secuencialmente las actividades de análisis, diseño, construcción, etc. y relacionan este ciclo con metodologías tradicionales como el PMI®.  Erróneamente también muchos practicantes de las metodologías ágiles creen que los ciclos de vida incrementales son exclusivos de estas. Ambos distan mucho de la realidad: hay evidencias de la definición y uso de ciclos de vida incrementales incluso en los años cincuenta. Un ejemplo es el ciclo de vida evolutivo en espiral de Boehm. Otro gran error es asumir que el ciclo de vida en cascada es absolutamente secuencial. Es muy difícil realizar todo el trabajo de requerimientos antes de iniciar el diseño de modo que dicha etapa se “cierre” y no haya necesidad de volver atrás. Creer en esto es negar que se vayan a presentar cambios a los requerimientos en etapas posteriores del desarrollo. El ciclo de vida en cascada secuencial data de 1957, sin embargo existen actualmente variantes del ciclo de vida en cascada (tal como la versión de Royce) que propone una visión incremental, reconociendo que el cambio es innegable y que por lo tanto siempre habrá que volver atrás para refinar sucesivamente los resultados de cada etapa de acuerdo con los cambios que se presentan.

En resumen ni cascada significa fracaso o arcaísmo, ni incremental significa éxito o ágil, cada equipo debe decidir cuál es el ciclo más conveniente para su proyecto. La elección del ciclo de vida del proyecto debe realizarse según las condiciones de cada proyecto.  Qué dice el PMBOK® al respecto ?  Primero mucha gente suele confundir los Procesos para la Dirección del proyecto con el Ciclo de vida de un proyecto.  Respecto de este último lo que menciona la Guía es lo siguiente: “El ciclo de vida del proyecto define las fases que conectan el inicio de un proyecto con su fin. La transición de una fase a otra dentro del ciclo de vida de un proyecto generalmente implica y, por lo general, está definida por alguna forma de transferencia técnica. Generalmente, los productos entregables de una fase se revisan para verificar si están completos, si son exactos y se aprueban antes de iniciar el trabajo de la siguiente fase. No existe una única manera, que sea la mejor, para definir el ciclo de vida ideal de un proyecto. Asimismo, las prácticas comunes de la industria a menudo conducen a usar un ciclo de vida preferido dentro de dicha industria”.

**La Necesidad de la Planificación:**Varios agilistas han coincidido en describir los procesos ágiles como “adaptables” en contraposición a los métodos tradicionales o “dirigidos por planes”. Hay malas interpretaciones sobre esto también. Los agilistas critican que se pretende planificar en excesivo detalle asumiendo que hay una capacidad de predicción que no es posible. Los tradicionalistas responden afirmando que no hay planificación en lo “ágil”. Son pocos los proyectos de software que se pueden predecir con mucho nivel de precisión, y por lo tanto lo más probable es que esas actividades se redefinan y el esfuerzo en planificación sea un desperdicio. Ningún método tradicional propone esto como buena práctica.

Por otro lado, es erróneo indicar que en los métodos ágiles no hay planificación. Todos incluyen prácticas específicas al respecto donde lo que varia sea probablemente la estrategia empleada y las técnicas utilizadas. En los métodos ágiles puede haber varios niveles de planificación: un plan de “releases” de alto nivel, un plan de iteraciones con un poco más de detalle y finalmente un plan de iteración donde aparecen las actividades concretas.

También es erróneo pensar que estas técnicas son exclusivas de las metodologías ágiles. Esta planificación gradual o “Rolling Wave Planning” es una técnica recomendada incluso por el PMBOK® donde se menciona explícitamente:  “la frecuencia de iteración de los procesos de planificación se ve afectada por cambios-  El plan de gestión del proyecto, desarrollado como una salida del Grupo de Procesos de Planificación, pondrá énfasis en la exploración de todos los aspectos del alcance, la tecnología, los riesgos y los costes. Las actualizaciones que surjan como consecuencia de cambios aprobados durante la ejecución del proyecto pueden causar un impacto significativo en partes del plan de gestión del proyecto. Las actualizaciones del plan de gestión del proyecto proporcionan más precisión respecto al cronograma, los costes y los requisitos de recursos a fin de satisfacer en su totalidad el alcance del proyecto definido. Las actualizaciones pueden limitarse a las actividades y puntos relacionados con la ejecución de una fase específica. Esta elaboración progresiva del plan de gestión del proyecto a menudo se denomina “planificación gradual” e indica que la planificación es un proceso repetitivo y continuo”.

La incertidumbre en los requerimientos y por lo tanto la necesidad del cambio ha sido aceptada desde siempre. En realidad las metodologías ágiles proponen reducir las actividades tipo “big up-front”. Por ejemplo, cuando el equipo tiene que dedicarse días enteros a actividades de diseño antes de codificar. En cambio se propone que el diseño sea una actividad permanente cuyos resultados se validan continuamente a la luz de lo construido. Según esto, el equipo dedica diariamente una parte de su tiempo a actividades de diseño, el cual evoluciona en paralelo con la construcción.

**La Necesidad de la Documentación:**Otro tema es la documentación. Aquí me resultó muy interesante y cómico un comentario que se le atribuye a Kent Beck el creador de Extreme Programming (XP) de una conversación que mantuvo con un practicante de su metodología:

Agradecido practicante: Sr. Beck, tengo que agradecerle, XP nos cambió la vida, sin embargo, ahora tenemos algunos problemas con el testing.   
Kent Beck: Muchas gracias. ¿Quiere decir que no logran codificar las pruebas antes de los programas?   
AP: Bueno, en realidad no codificamos pruebas…   
KB: Entiendo, ¿entonces logran realizar integración frecuente generando una nueva versión funcional al menos una vez al día?    
AP: Mmm, en realidad no…    
KB: ¿Pero funcionan la programación en pares, las historias de usuario o el refactoring?    
AP: Mmm, no…   
KB: Entonces, ¿cómo es que hacen XP?    
AP: Ah, bueno, dejamos de documentar…

Beck ejemplificaba de esta forma exagerada lo que estaba dándose en la realidad: las nuevas metodologías fueron acogidas por muchos practicantes con gran entusiasmo y poco criterio. También es cierto que muchos practicantes de metodologías tradicionales han implementado sus procesos con dependencia extrema de los documentos, formalismos y rigurosidades para cualquier tipo de proyecto sin considerar tamaño, complejidad o duración. Malas interpretaciones pueden llevar a que una compañía tenga un estándar de plan de proyecto de miles de páginas y lo exija para todos sus proyectos sin importar si uno es de 50 horas hombre y otro de 5.000. También he visto como compañías exigen a sus equipos documentar el diseño usando todos los diagramas de UML, sin distinción de cuáles son realmente útiles para cada proyecto o situación.Si bien varios autores han asumido posiciones relativamente extremas sobre el asunto de documentación, en general hay consenso en que cada equipo debe determinar la cantidad y forma de documentación requerida. Además, deben tomarse en cuenta los requerimientos del cliente y condiciones debidas a regulaciones o estándares. Cumplir con los requerimientos de documentación y formalidades como control de cambios y riesgos no significa que el equipo deje de ser “ágil”.  Finalmente la documentación no reemplaza la comunicación.

**La Necesidad de un Balance:** Son muchos los temas que dan lugar a debate y polémica durante los distintos cursos y seminarios sobre metodologías de administración de proyectos: utilización de casos de pruebas, manejo de riesgos, CMMI y Prince2, cual es el rol del PM en métodos ágiles, etc. son algunos de ellos.  Probablemente la polémica continuará por algún tiempo más.  Hoy, sin embargo, se ha visto la necesidad de encontrar un balance. Las prácticas ágiles y las más tradicionales han demostrado tener nichos en los cuales se comportan bien y favorecen el éxito.  Antes podía decirse que estos nichos eran independientes y un equipo podía escoger entre ir ágil o tradicional. Lo que sucede cada vez con mayor frecuencia en el vertiginoso mundo en que vivimos, es que estos nichos se entrecruzan frecuentemente, y por lo tanto ningún esquema resulta completo o favorable por sí sólo.No hay que quedarse en ningún extremo. Eso puede llevarnos a ignorar los objetivos del proyecto y conducirnos ciegamente al fracaso. Por el contrario, reconociendo el valor que una mezcla adecuada puede generar, aumentaremos notablemente la probabilidad de éxito de nuestros proyectos no sólo en los parámetros usuales de plazo, alcance y costo, sino también en el desarrollo del conocimiento y experiencia para el equipo y la organización.

**Conclusiones:**La metodología ágil de proyectos se utiliza básicamente para proyectos de IT y dentro de ellos esencialmente para el desarrollo de software.  Surgieron como respuesta a la dificultad de acomodar cambios, requerimientos no claros, entregas rápidas y adaptación. Las prácticas ágiles no están limitadas al uso de Scrum, XP, FDD u otra “marca” de métodos ágiles. En nuestro curso IT Project Management explicamos como se mapean muchas prácticas ágiles con un método tradicional como el PMBOK.  Como simplificación podriamos decir que utilizando estas 3 técnicas estamos de alguna manera agilizando nuestro proceso:

* Desarrollo Iterativo-Incremental: cada iteración del SDLC sigue el mismo proceso de desarrollo, entrega algo más de valor al cliente en etapas y espera su aprobación para la siguiente iteración, permite introducir cambios o mejoras al final de cada iteración, el cliente y el team colaboran para determinar el alcance de cada iteración (historias de usuario que definan la funcionalidad, lista de “features” agrupados, selección de requerimientos de una backlog list, etc). Esta aproximación de desarrollo apunta a otro concepto importante: el “time Boxing”, donde el PM pregunta cuanto del proyecto (en términos de funciones) puede ser desarrollado en un período de tiempo pre-establecido (normalmente corto: dos o cuatro semanas). Cada desarrollo incremental otorga al cliente un paso más para llegar al producto final.
* Equipos de alto rendimiento que trabajan junto con el cliente: muchas prácticas ágiles incluyen al cliente en fases tempranas del desarrollo y está siempre representado dentro del equipo de trabajo (on site, experto, product owner, etc.). Esta técnica (incluír al ciente dentro del equipo de trabajo) ofrece a menudo mucha resistencia dado que los clientes no quieren dedicar recursos full-time al desarrollo. Es cuestión de hacerle entender al cliente que la calidad del producto final se incrementa en proporción al tiempo que el cliente dedica su tiempo al team de desarrollo.
* Mantener el desarrollo simple: un desarrollo ágil trata de que el proceso sea simple, el código menor posible, entregar solo lo que sea necesario y documentar solo lo imprescindible.  Se utiliza mucho la práctica de “refactoring” para simplificar el código y eliminar redundancias. Otra opción de mantener las cosas simples es la práctica de Lean Project Management[[1]](https://pmqlinkedin.wordpress.com/wp-admin/page-new.php#_ftn1) que básicamente consiste en emplear las técnicas de Lean Manufacturing, Six Sigma y TOC en la gestión de proyectos (herramientas de control estadístico, procesos ágiles o rápidos y eficientes, value stream mapping, uso de cadena crítica, eliminación de desperdicios y utilización de los principios de “Poka Yone” entre otros.

[LinkedIn](https://www.quonext.com/#linkedin)[Twitter](https://www.quonext.com/#twitter)[Facebook](https://www.quonext.com/#facebook)[Google+](https://www.quonext.com/#google_plus)[WhatsApp](https://www.quonext.com/#whatsapp)[Email](https://www.quonext.com/#email)[Compartir](https://www.addtoany.com/share#url=https%3A%2F%2Fwww.quonext.com%2Fblog%2Fmetodologias-agiles-scrum-kanban-xp%2F&title=Metodolog%C3%ADas%20%C3%81giles%20en%20el%20desarrollo%20de%20software)

# **Metodologías Ágiles en el desarrollo de software**

Las **metodologías ágiles** son un conjunto de técnicas para gestionar y **desarrollar proyectos de software** donde los requisitos evolucionan para adaptarse a las necesidades del proyecto. Algunos de los rasgos comunes entre las varias **metodologías ágiles** que detallaremos en este post son la capacidad de los equipos para auto-organizarse, la polivalencia de los miembros del equipo desarrollador y la toma de decisiones a corto plazo.

Estas **metodologías ágiles** pretenden mejorar tanto la productividad del conjunto del equipo como la calidad final del desarrollo.

### ¿Qué problemas tienen las técnicas clásicas?proyectos-ingenieria-software

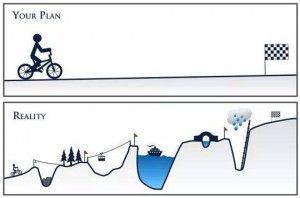
* Planificaciones poco realistas
* Retrasos, falta de tiempo para testeos
* Cambios de contexto y alcance
* Cliente poco involucrado
* Falta de comunicación
* Falta de motivación del equipo desarrollador
* Falta de flexibilidad
* Resultado no es el esperado por el cliente

Estos problemas pueden causar un desperdicio de la inversión en dinero y tiempo. Además las probabilidades de fracaso son mayores si existen este tipo de dificultades durante el desarrollo y ninguna de las partes estará satisfecha, ni el equipo desarrollador ni el cliente.

### Prioridades del manifiesto Ágil

* Considerar más importantes a los individuos e interacciones que a los procesos y herramientas
* Sobreponer que el software funcione a la documentación excesiva
* Colaborar con el cliente a la negociación contractual
* Priorizar la respuesta ante los cambios a seguir con un plan

## **Scrum**



Scrum es un marco de trabajo para los desarrollos ágiles. Sus principales características son:

* Adoptar una estrategia de desarrollo incremental, se planifican varios bloques temporales (de dos semanas o 1 mes, por ejemplo) llamados iteraciones. Cada iteración incluye: planificación, análisis de requisitos, diseño, codificación, pruebas y documentación. De de manera incremental, se irán completando los requisitos.
* Priorización, estos requisitos se irán cumpliendo dando prioridad primero a los que aportes más valor al cliente. Así conseguimos descargar presión a los desarrolladores del proyecto y el cliente recibirá los resultados más importantes en las primeras iteraciones.
* **Equipos auto organizados**
* Desarrollo de fases de forma paralela. Si las fases del proyecto son independientes, avanzar en más de una fase a la vez acortará los tiempos del proyecto.
* **Mejora continua.**

## **Kanban**



La metodología **Kanban** es complementaria a **Scrum.** En **Kanban** se visualiza el flujo de trabajo y este se tiene que dividir en tareas e incluirlas en el tablero.

Organizaremos el **tablero de Kanban** en:

* Tareas que hay que hacer
* Tareas en curso
* Tareas terminadas

#### **Principios de Kanban**

1. **Visualizar el flujo de trabajo**: esquematizar el flujo de trabajo para que de manera visual se puedan ver las oportunidades de mejora fácilmente.
2. **Limitar el trabajo en progreso**: limitar la cantidad de trabajo para que se pueda realizar y administrar de manera razonable.
3. Respetar los **roles** y mantener un **liderazgo** en todos los niveles.
4. **Seguimiento, monitoreo y análisis** constantes para buscar formas de mejoras allanando el terreno para futuras actualizaciones.

## **XP – Extreme Programming**

Las técnicas de **XP** han de permitir flexibilidad y cambios en los requisitos, además de fomentar la simplicidad, comunicación, trabajo en equipo, retroalimentación y respeto. Algunos ejemplos de estas técnicas:

* **TDD**: Desarrollo guiado por pruebas. Estas pruebas deben ser escritas antes del desarrollo para validar las funcionalidades.
* **Pair Programming**: Técnica en la que dos programadores comparten ordenador para programar a la vez.
* **Integración con cliente**: Una persona del cliente debe trabajar junto al equipo de desarrollo, así mejora la comunicación y el seguimiento de cambios solicitados por el cliente.
* **Refactorización**: Repasar y sobrescribir ciertas partes del código para mantenerlo lo más limpio y legible.
* **Propiedad compartida**: Promover que todos los miembros sean capaces de tocar cualquier parte del código.
* **Simplicidad**: Cuanto más simple sea el sistema construido más sencillo será comprenderlo, modificarlo o añadir nuevas funcionalidades.

## **¿Las metodologías ágiles son compatibles?**

Todas las **metodologías ágiles** pueden ser complementarias entre sí, al menos de forma parcial:

La **metodología Scrum** está pensada para aumentar la productividad, y **XP** para agregar valor y calidad al proyecto. Por esto muchas organizaciones que utilizan Scrum adoptan el Pair Programming, el TDD y la refactorización como prácticas que mejoran calidad, consiguiendo al mismo tiempo una productividad alta.

Por otro lado, a menudo se utiliza el **tablero de Kanban** por los **equipos de Scrum** para realizar un seguimiento de las tareas que hay en cada sprint.

modelo de cascada  
En [Ingeniería de software](http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software) el **desarrollo en cascada**, también llamado **modelo en cascada**, es el enfoque metodológico que ordena rigurosamente las etapas del [**proceso para el desarrollo de software**](http://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software), de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior.[1](http://es.wikipedia.org/wiki/Desarrollo_en_cascada#cite_note-1)

Un ejemplo de una metodología de desarrollo en cascada es:

1. Análisis de requisitos.
2. Diseño del Sistema.
3. Diseño del Programa.
4. Codificación.
5. Pruebas.
6. Implantación.
7. Mantenimiento.

De esta forma, cualquier error de diseño detectado en la etapa de prueba conduce necesariamente al rediseño y nueva programación del código afectado, aumentando los costos del desarrollo. La palabra *cascada* sugiere, mediante la metáfora de la fuerza de la gravedad, el esfuerzo necesario para introducir un cambio en las fases más avanzadas de un proyecto.

Si bien ha sido ampliamente criticado desde el ámbito académico y la industria, sigue siendo el paradigma más seguido al día de hoy.

**Análisis de requisitos**

En esta fase se analizan las necesidades de los usuarios finales del software para determinar qué objetivos debe cubrir. De esta fase surge una memoria llamada SRD (documento de especificación de requisitos), que contiene la especificación completa de lo que debe hacer el sistema sin entrar en detalles internos.

Es importante señalar que en esta etapa se debe **consensuar** todo lo que se requiere del sistema y será aquello lo que seguirá en las siguientes etapas, no pudiéndose requerir nuevos resultados a mitad del proceso de elaboración del software.

**Diseño del Sistema**

Descompone y organiza el sistema en elementos que puedan elaborarse por separado, aprovechando las ventajas del desarrollo en equipo. Como resultado surge el SDD (Documento de Diseño del Software), que contiene la descripción de la estructura relacional global del sistema y la especificación de lo que debe hacer cada una de sus partes, así como la manera en que se combinan unas con otras.

Es conveniente distinguir entre diseño de alto nivel o arquitectónico y diseño detallado. El primero de ellos tiene como objetivo definir la estructura de la solución (una vez que la fase de análisis ha descrito el problema) identificando grandes módulos (conjuntos de funciones que van a estar asociadas) y sus relaciones. Con ello se define la arquitectura de la solución elegida. El segundo define los algoritmos empleados y la organización del código para comenzar la implementación.

**Diseño del Programa**

Es la fase en donde se realizan los algoritmos necesarios para el cumplimiento de los requerimientos del usuario así como también los análisis necesarios para saber que herramientas usar en la etapa de Codificación.

**Codificación**

Es la fase en donde se implementa el [código fuente](http://es.wikipedia.org/wiki/C%C3%B3digo_fuente), haciendo uso de prototipos así como de pruebas y ensayos para corregir [errores](http://es.wikipedia.org/wiki/Error_de_software).

Dependiendo del lenguaje de programación y su versión se crean las bibliotecas y componentes reutilizables dentro del mismo proyecto para hacer que la programación sea un proceso mucho más rápido.

**Pruebas**

Los elementos, ya programados, se ensamblan para componer el sistema y se comprueba que funciona correctamente y que cumple con los requisitos, antes de ser entregado al usuario final.

**Verificación**

Es la fase en donde el usuario final ejecuta el sistema, para ello el o los programadores ya realizaron exhaustivas pruebas para comprobar que el sistema no falle.

En la creacion de desarrollo de cascada se implementa los codigos de investigacion y puebas del mismo

Mantenimiento

Una de las etapas mas criticas, ya que se destina un 75% de los recursos, es el mantenimiento del Software ya que al utilizarlo como usuario final puede ser que no cumpla con todas nuestras expectativas.

imagen

## http://ingsw.pbworks.com/f/1247321922/modelo-cascada.JPG



modelo o desarrollo espiral

El **desarrollo en espiral** es un [modelo](http://es.wikipedia.org/wiki/Modelo_matem%C3%A1tico) de [ciclo de vida del software](http://es.wikipedia.org/wiki/Ciclo_de_vida_del_software) definido por primera vez por [Barry Boehm](http://es.wikipedia.org/wiki/Barry_Boehm) en 1986, utilizado generalmente en la [Ingeniería de software](http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software). Las actividades de este modelo se conforman en una [espiral](http://es.wikipedia.org/wiki/Espiral), en la que cada bucle o [iteración](http://es.wikipedia.org/wiki/Iteraci%C3%B3n) representa un conjunto de actividades. Las actividades no están fijadas a ninguna prioridad, sino que las siguientes se eligen en función del [análisis de riesgo](http://es.wikipedia.org/wiki/An%C3%A1lisis_de_riesgo), comenzando por el bucle interior.

La Ingeniería de software, se vale y establece a partir 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 concebido fue el de Royce, más comúnmente conocido como [*desarrollo en cascada*](http://es.wikipedia.org/wiki/Desarrollo_en_cascada) o *desarrollo lineal secuencial*. Este modelo establece que las diversas actividades que se van realizando al desarrollar un producto software se suceden de forma lineal.

Boehm, autor de diversos artículos de ingeniería del software; modelos de estimación de esfuerzo 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 riesgo más asumible y se hace un ciclo de la espiral. Si el cliente quiere seguir haciendo mejoras en el software, se vuelve a evaluar las distintas 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 fue propuesto por [Boehm](http://es.wikipedia.org/wiki/Barry_Boehm) en [1988](http://es.wikipedia.org/wiki/1988). 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 Cascada, pero no necesariamente debe ser así. El Espiral puede verse como un modelo evolutivo que conjuga la naturaleza iterativa del modelo [MCP](http://es.wikipedia.org/wiki/Modelo_de_prototipos) con los aspectos controlados y sistemáticos del [Modelo Cascada](http://es.wikipedia.org/wiki/Modelo_Cascada), con el agregado de gestión de riegos.

ciclos

En cada vuelta o iteración hay que tener en cuenta:

* **Los Objetivos:** qué necesidad debe cubrir el producto.
* **Alternativas:** las diferentes formas de conseguir los objetivos de forma exitosa, desde diferentes puntos de vista como pueden ser:

1. **Características:** experiencia del personal, requisitos a cumplir, etc.
2. **Formas de gestión del sistema**.
3. **Riesgo asumido con cada alternativa**.

* **Desarrollar y Verificar:** Programar y probar el software.

Si el resultado no es el adecuado o se necesita implementar mejoras o funcionalidades:

* Se planificaran los siguientes pasos y se comienza un nuevo ciclo de la espiral. La espiral tiene una forma de [caracola](http://es.wikipedia.org/wiki/Caracola) y se dice que mantiene dos dimensiones, la radial y la angular:

1. **Angular:** Indica el avance del proyecto del software dentro de un ciclo.
2. **Radial:** Indica el aumento del coste del proyecto, ya que con cada nueva iteración se pasa más tiempo desarrollando.

Este sistema es muy utilizado en proyectos grandes y complejos como puede ser, por ejemplo, la creación de un Sistema Operativo.

Al ser un modelo de Ciclo de Vida orientado a la gestión de riesgo se dice que uno de los aspectos fundamentales de su éxito radica en que el equipo que lo aplique tenga la necesaria experiencia y habilidad para detectar y catalogar correctamente los riesgos.

ejemplo

