Universidad Latinoamericana de Ciencia y Tecnología
ULACIT

Facultad de Ingeniería

Escuela de Ingeniería Informática

Trabajo final para optar por el grado de Licenciatura en Informática con énfasis en Gestión de Recursos Tecnológicos

Tema

¿Cómo implementar las metodologías ágiles en las empresas de desarrollo de software?

Sustentante: Jorge Mauricio Acuña Gómez

Cédula: 1-1060-241

Tutor: Lic. Miguel Pérez Montero

3 Cuatrimestre 2009

Tabla de contenido

Referencias de ilustraciones

Introducción

Hoy la tecnología es considerada un requisito para que las empresas u organizaciones trabajen de una manera competitiva. Dentro de esa tecnología podemos hablar del desarrollo de software. Sabemos que la automatización de procesos es de suma importancia para llevar a cabo nuestro trabajo de una manera más fácil, exacta y eficiente, lo que lleva a obtener buenos resultados.

Existen varias metodologías cuando hablamos de desarrollar software, pero la mayoría nos envuelven en procesos burocráticos y tediosos, además de que los resultados o la funcionalidad real del producto, (en este caso el software desarrollado) se van viendo a un mediano o largo plazo.

Las metodologías ágiles o procesos ágiles de desarrollo de software como también se les llama, intentan evitar esos procesos burocráticos y tediosos que comentábamos anteriormente, enfocando sus esfuerzos en la gente y en los resultados a corto plazo por medio de iteraciones donde cada fase o propuesta será evaluada con detalle. De esta manera se podrá contar con un pequeño producto funcional al final de cada revisión o iteración como se le llama a cada una de estas revisiones.

Cuando se habla de proyectos de software, muchas veces se dice que una de las causas del fracaso es por un mal análisis de requerimientos o porque durante el proyecto es casi al final cuando se va obteniendo un producto funcional.

Esta es una de las razones por la cual la implementación de metodologías ágiles ayudará a minimizar este gran problema que las empresas o departamentos dedicados al desarrollo de software sufren o viven en la actualidad.

¿Qué es el *Agile Manifesto*? [1]

Según la página sobre el *agile manifesto* (Agile Manifesto, 2001) podemos definir el manifiesto para el desarrollo de software de la siguiente manera:

> Estamos descubriendo mejores formas de desarrollar software haciéndolo y ayudando a otros a hacerlo.
> A través de este trabajo hemos llegado a valorar:
>
> - Los individuos y las interacciones son más importantes que los procesos y herramientas.
> - Software funcional es más importante que la documentación exhaustiva.
> - La colaboración con el cliente es más importante que la negociación del contrato.
> - La respuesta al cambio es más importante que el seguimiento de un plan.

El *agile manifesto* centra su importancia en el individuo como tal y en su interacción como parte fundamental dentro del desarrollo de un proyecto de software, ya que la persona como miembro de un equipo de trabajo es la encargada de proponer nuevas ideas y llevarlas a cabo. La creatividad y proactividad es de mucho valor a la hora de implementar esta metodología.

Anteriormente las metodologías daban mucha importancia a la documentación como tal. Tener una buena documentación es importante pues ayuda a comprender el funcionamiento de las aplicaciones y la relación o integración que existe con los diferentes componentes que las conforman, pero es mucho mejor contar con herramientas o sistemas realmente funcionales y darles prioridad antes que a la documentación.

Es primordial contar con un software funcional antes de realizar su documentación. Muchas veces en los proyectos se le da más importancia a la documentación, incluso a contar con requerimientos debidamente implementados y probados.

Otro aspecto importante es el cliente o el usuario final del sistema que se va a desarrollar. El usuario final es el usuario experto y quien mejor conoce del proceso, por eso es de suma importancia contar con él como parte fundamental durante todo el ciclo de vida del desarrollo del sistema. Como usuario experto, él nos puede orientar e incluso proponer nuevas ideas o formas de llevar a cabo los procesos a implementar o automatizar.

[1] Consultado de www.agilemanifesto.org

Respecto a la respuesta al cambio, el equipo de trabajo debe ser flexible ante los cambios que se puedan presentar durante cualquier etapa del ciclo de vida. Debemos tener la premisa de que uno de los factores constantes puede ser precisamente el cambio.

Los 12 principios del *Agile Manifesto* [2]

Según la página Web del *agile manifesto* (Agile Manifesto, 2001), podemos citar los 12 principios de la siguiente manera:

1. Nuestra máxima prioridad es satisfacer al cliente través de la entrega oportuna y continua de software de valor.

2. Los cambios de requerimientos son bienvenidos, incluso en las etapas finales del desarrollo. Los procesos ágiles aprovechan el cambio para dar ventaja competitiva al cliente.

3. Entrega de software funcional con frecuencia, desde semanas hasta meses, preferiblemente en periodos cortos de tiempo.

4. La gente de negocios y desarrolladores deben trabajar juntos a diario durante todo el proyecto.

5. Construir proyectos en torno a individuos motivados. Darles el medio ambiente y el apoyo que necesitan, y confiar en ellos para hacer el trabajo.

6. El método más eficiente y efectivo de transmitir o transferir la información de un desarrollo es por medio de las reuniones cara a cara o uno a uno.

7. El software funcional es la principal medida de progreso.

8. Los procesos ágiles promueven el desarrollo sostenible. Los patrocinadores, desarrolladores y usuarios deben poder mantener un ritmo constante indefinidamente.

9. La atención continua a la excelencia técnica y el buen diseño mejora la agilidad.

10. Simplicidad - el arte de maximizar la cantidad de un trabajo no realizado - es esencial.

[2] Consultado de www.agilemanifesto.org

11. Las mejores arquitecturas, requisitos y diseños surgen de la auto-organización de los equipos.

12. En intervalos regulares de tiempo el equipo reflexiona sobre cómo se puede mejorar y ser más efectivo y luego se tratan de corregir y ajustar las acciones de acuerdo con lo que vaya sucediendo, con el fin de mejorar los procesos.

Como podemos ver, las metodologías ágiles o el *agile manifesto* no dicen cómo implementar un proceso; dan sugerencias de cómo se pueden mejorar los procesos basados principalmente en una serie de principios.

De tal manera, se puede concluir que la implementación de las metodologías ágiles se debe realizar básicamente tomando estos principios como guía y teniendo en cuenta el sentido común.

¿Por qué el sentido común?

Por un lado porque todas las organizaciones, empresas o instituciones son diferentes, ya que tienen negocios, situaciones, clientes y gente diferentes.

Entonces, la mejor manera de implementarlo es seguir el sentido común y ver cuáles mecanismos, métodos o procesos existentes pueden ayudar a cumplir con los principios establecidos por el *Agile Manisfesto*.

Por otro lado, no necesariamente tienen que ser métodos o metodologías existentes, pueden ser creadas en la misma compañía, siempre y cuando ayuden a mejorar y a cumplir con los principios del *Agile Manifesto*.

De la misma forma, por ser un asunto dependiente del negocio y la organización pudiera ser que algunos de los principios se implementen y otros no. No necesariamente todos deben ser implementados.

Todo cambio es duro. En las empresas la organización y la mejora de procesos algunas veces es un proceso tedioso y difícil. Estas metodologías impulsan al cambio, a la mejora, a la organización y documentación, por lo que se hace difícil el proceso de implementación.

Lo ideal para iniciar con el cambio en una organización es llevar a cabo retrospectivas con los equipos de trabajo y la mejor forma de realizarlas es respondiendo a las siguientes preguntas:

- ¿Qué hemos hecho bien?
- ¿Qué hemos hecho mal?
- ¿Cómo podemos mejorar?

Realizar esto periódicamente ayuda a encontrar las debilidades y fortalezas como equipo de trabajo. De esta manera se logra tener un buen punto de partida, además que con este tipo de reuniones se cumple también con uno de los principios de la metodología, que es la transferencia de información por medio de reuniones cara a cara.

Así podemos saber qué mejorar y es ahí donde empezamos a buscar qué principios pueden ayudar a mejorar las debilidades detectadas en nuestros procesos y equipos de trabajo.

Existen muchas formas o sugerencias de cómo implementar y llevar a cabo el cumplimiento de estos principios, como lo son:

- *Cleanroom*
- DSDM (*Dynamic Systems Development Method*)
- Modelo Iterativo
- XP (eXtreme Programming)
- *Scrum*
- *LeanSoftware Development*
- *FDD (Feature Driven Development)*
- *RUP (Rational Unified Process)*
- *V-Model*
- *RAD (Rapid Application Development)*
- *TDD (Test Driven Development)*

La práctica por realizar en este punto es analizar cada una de ellas y ver cuál o cuáles de ellas pueden ayudar a mejorar. De igual forma se pueden tomar las mejores prácticas de cada una y discutirlas en el equipo de trabajo, ajustarlas de acuerdo con las necesidades e implementarlas, pues el objetivo principal es cumplir con los principios. *Agile* no es una receta de mecanismos o procesos que debemos seguir “al pie de la letra”, son objetivos que debemos cumplir.

En el presente trabajo se comentarán las principales metodologías aplicadas actualmente para cumplir con los principios del *agile manisfesto.*

***Cleanroom software engineering* (Mills, H.D.; Dyer, M.; Linger, R.C, IEEE 1987)**

Es un proceso de desarrollo de software con el propósito de producir *software* con un nivel certificado de confiabilidad y credibilidad. El objetivo de este proceso es evitar los defectos en el *software* en vez de eliminar o corregir los defectos encontrados.

Los principios básicos de este proceso son:

1. Desarrollo de *software* basado en métodos formales (técnicas matemáticas basadas en la especificación, desarrollo y verificación de software y hardware de sistemas).

2. Basado en la implementación progresiva de la solución o producto final mediante iteraciones en el tiempo y evaluar con detalle su avance.

3. El incremento de la funcionalidad implementada será gradual.

4. La calidad de cada incremento se mide con estándares preestablecidos para verificar que el proceso de desarrollo se está llevando a cabo aceptablemente.

5. El incumplimiento de alguno de los estándares de calidad establecidos llevará al proyecto nuevamente a la fase de análisis.

6. Se debe incluir una fase de *testing* donde se pondrán a prueba las funcionalidades desarrolladas en el *software.* Se debe realizar una especificación formal de un subconjunto representativo de datos de entrada y salida con el fin de probar el *software*. De esta manera se puede su fiabilidad.

Dynamic Systems Development Method (DSDM Consortium, 2006).

DSDM es un método iterativo e incremental que hace hincapié en la participación continua del usuario final.

Su objetivo es entregar software a tiempo y ajustar o realizar cambios de acuerdo con las necesidades a lo largo del proceso de desarrollo.

DSDM se enfoca en proyectos caracterizados por tener un calendario apretado y poco presupuesto.

Los principios de este método son los siguientes:

1. La participación de los usuarios en la clave principal en la gestión de un proyecto eficiente y eficaz, donde los usuarios y desarrolladores comparten un lugar de trabajo, de manera que las decisiones podrán tomarse con mayor exactitud.

2. El equipo del proyecto debe estar facultado para tomar decisiones importantes para el progreso del proyecto sin esperar por la aprobación de nivel superior.

3. Enfoque en la entrega frecuente de productos, teniendo claro el supuesto de que siempre es mejor ir entregando algo "suficientemente bueno" que entregar algo "perfecto" al final. Con la entrega gradual de productos en el tiempo nos garantizamos la prueba de las nuevas funcionalidades implementadas, quedando el documento de revisión de las pruebas como punto de arranque para la siguiente iteración.

4. Los principales criterios para la aceptación de un entregable es la entrega de un sistema que responda a las necesidades empresariales actuales. La entrega de un sistema perfecto que se ocupa de todas las posibles necesidades de negocio es menos importante que centrarse en las funciones críticas.

5. El desarrollo es iterativo e incremental, y gracias a las sugerencias de los usuarios es que podemos llegar a una solución de negocio efectiva.

6. Todos los cambios durante el desarrollo son reversibles.

7. El alcance del proyecto y requerimientos deben ser definidos previamente y ser la base para su inicio.

8. La fase de *testing* o pruebas es llevada a cabo durante todo el ciclo de vida del proyecto.

9. Para ser eficientes y eficaces se requiere de la comunicación y cooperación entre todos los interesados en el proyecto.

Modelo de desarrollo iterativo

Este modelo de desarrollo de software cíclico fue desarrollado como respuesta a las debilidades del modelo en cascada. Se inicia con una planificación inicial y termina con el despliegue de la interacción cíclica entre las fases.

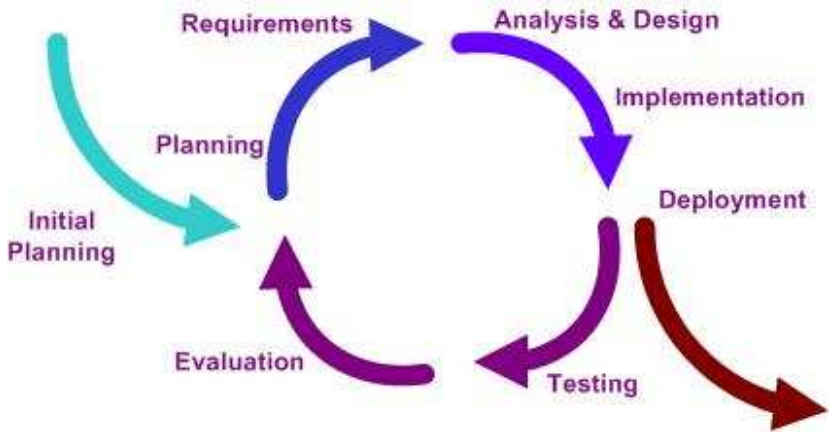

Figura 1
Modelo de desarrollo de software iterativo
Tomado de
http://en.wikipedia.org/wiki/Iterative_and_incremental_development

Guías de implementación

1. Cualquier dificultad en el diseño, codificación o pruebas de un requerimiento señalan la necesidad de rediseñar o re-codificar.

2. Las modificaciones en el código o tablas deben ser fáciles de mantener; si no es así es necesario rediseñar y recodificar.

3. Los parches pueden ser necesarios solo para evitar el rediseño durante una fase de ejecución.

4. La implementación del proyecto debe ser analizada frecuentemente para determinar qué tan bien se están cumpliendo sus objetivos.

5. La participación del usuario final es muy importante.

eXtreme Programming

Este modelo es uno de los más interesantes y usados y se basa en un conjunto de valores y buenas prácticas, donde el objetivo es incrementar la productividad a la hora de desarrollar sistemas o aplicaciones.

Al usar XP se debe esperar lo siguiente:

1. Los trabajos que dan resultado directo tienen prioridad.
2. Se reduce la burocracia que existe alrededor de la programación.
3. Se busca la simplicidad a la hora de desarrollar.
4. Usar el sentido común.

Los principios básicos de esta metodología son:

1. Se debe definir un conjunto de pruebas o escenarios, donde se establecen las entradas y resultados esperados, productos de los datos de entrada. Generalmente para cumplir con este principio se automatizan estos escenarios por medio de *Unit tests.*

2. La planificación de las actividades y tiempos de duración de cada uno de ellas se llevará a cabo basándose en un documento creado por el usuario llamado Historias del usuario (se deben tener entre 20 y 80 historias), donde él mismo definirá sus necesidades a través de las diferentes actividades que realizará en el sistema.

   A partir de este documento creado por el usuario se crea otro documento llamado Plan de liberación, en el cual se definen los tiempos de entrega de la aplicación y la retroalimentación por parte del usuario final.

   En la fase de planificación es recomendable la interacción diaria con todo el equipo y el usuario, con el fin de ir proponiendo soluciones y resolviendo problemas e inquietudes que se presentan a lo largo de esta etapa.

3. La interacción con el usuario final es de suma importancia, por lo que se le da poder absoluto para:

   a. Definir requerimientos.
   b. Definir prioridades.
   c. Responder preguntas de los programadores.

   Esta interacción con el usuario disminuye la documentación, el tiempo de comunicación y los costos de creación y mantenimiento, por lo que aumenta la eficiencia del proceso como tal.

4. En este modelo no existe la fase inicial de requerimientos; en vez de eso se permite que los requerimientos se vayan definiendo durante el desarrollo del proyecto. Por eso es de suma importancia el rol del usuario final. El usuario es parte fundamental para la definición de los requerimientos, por eso debe estar siempre disponible para cualquier consulta que los desarrolladores puedan tener en cualquier momento

5. Uno de los principios más radicales propuestos por XP es la programación en parejas. Este punto es uno de las características más importantes que definen a XP pues establece que todos los desarrollos deben ser realizados por parejas de programadores y en una sola computadora compartida.

   Es importante recalcar que la idea de la programación en parejas es buscar más de una solución a problemas o defectos que puedan ir apareciendo durante el desarrollo, además de las propuestas de como desarrollar los

requerimientos propuestos por el usuario final. Así mismo, se puede transmitir la experiencia y conocimiento por parte del programador más experimentado, de esta manera también se garantiza la capacitación y de esta manera ir fortaleciendo nuestros equipos de trabajo.

6. La creación de versiones de *builds* estables no se hace basada en actividades de cronograma establecidos, sino que se realizan tantos como se requieran. Se pueden generar varias versiones al día de ser necesario.

   Esto reduce el tiempo para tener *builds* funcionales, pues los problemas que puedan surgir en una determinada versión del sistema pueden ser resueltos rápidamente en la siguiente versión del *build.* Es decir, no hay reglas ni restricciones sobre cuándo se debe generar un *build.*

7. Se debe contar con espacio para la refactorización. Este proceso debe ser constante durante y hasta el final del proyecto.

   Los procesos desarrollados y a desarrollar deben ser evaluados constantemente y recodificados de ser necesario, con el fin de siempre buscar la mejora y la eficiencia de los sistemas o aplicaciones.

8. Se deben ir haciendo *deployment* del sistema en producción constantemente, de esta manera en períodos cortos se va a hacer entrega de software funcional, además de contar con la posibilidad de probar las diferentes versiones del sistema en ambientes de producción y datos reales.

9. La solución propuesta siempre debe ser las más sencilla y siempre cumpliendo con todas las necesidades del usuario final.

   Al contar con soluciones simples se garantiza un entendimiento fácil del sistema y un mantenimiento fácil de realizar.

10. Antes de iniciar el proyecto se define una idea integral de cómo va a funcionar el sistema. Se definen breves descripciones de los posibles escenarios del sistema. No se usa UML para definirlos.

    A esta idea inicial se le llama “Metáfora”. La solución o escenario debe ser comparada con alguna situación real, de esta manera se logra el entendimiento por parte de todos los integrantes del equipo de trabajo.

    Se definen tarjetas CRC (clase, responsabilidad y colaboración) en donde se determinan las clases que se van a trabajar, quien es el responsable y la colaboración que hay con las otras clases (cómo se relacionan o integran unas con otras).

Esto va a ayudar definitivamente a la hora de establecer actividades, prioridades y tiempos de desarrollo.

11. El código es propiedad compartida. Todos son dueños de todo.

12. Se debe definir previamente un estándar de codificación. Este estándar será utilizado por todos los programadores del proyecto.

13. Programadores cansados no producen código de calidad, por lo que XP busca evitar o disminuir las horas o trabajo extra. Se deben cumplir las 40 horas semanales.

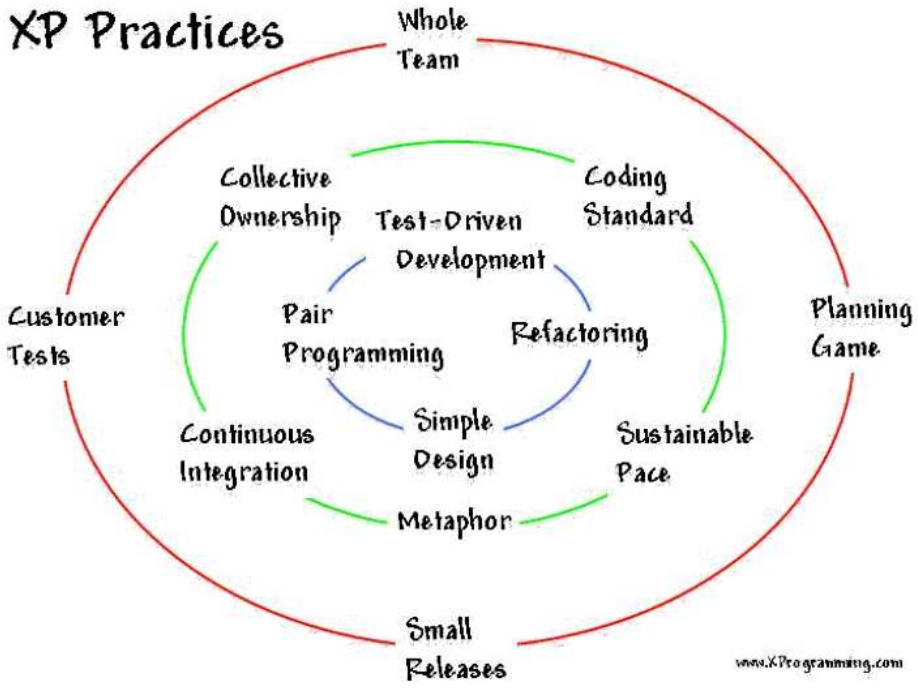

Figura 2
Modelo *eXtreme Programming*
Tomada de www.XProgramming.com

Scrum [3]

Es una metodología aplicada a la gestión de proyectos que busca maximizar la productividad de un equipo de trabajo. De la mima forma busca disminuir la burocracia y las actividades no orientadas a producir software funcional.

A la hora de implementar *Scrum* debemos tomar en cuenta los siguientes principios:

1. Se intenta obtener resultados en períodos muy cortos (cada 30 días aproximadamente). Se hacen entregas parciales y regulares del resultado final del proyecto. La entrega de estos resultados parciales debe ser lo más pronto posible.

2. Se trata más sobre la gestión y administración del proyecto de software que sobre el desarrollo o la programación, pues para esto tenemos otras metodologías como lo es *eXtreme Programming, por ejemplo.*

3. Busca delegar la responsabilidad en el equipo para tomar la decisión que más convenga para llevar a cabo una determinada actividad, con el objetivo de ser lo más productivos posibles como se mencionó anteriormente.

4. Innovación, competitividad y producción son fundamentales.

5. Los proyectos deben ejecutarse en bloques temporales, cortos y fijos en iteraciones de un mes o cada 2 semanas, si así se requiere.

6. Producto de cada iteración se debe obtener un resultado completo, un incremento del producto final que sea significativo y "palpable".

7. Al inicio se definen los requisitos donde el cliente se encarga de priorizar cada uno de ellos quedando repartidos en iteraciones y puntos de entrega.

8. *Scrum* se compone de las siguientes actividades:

    a. ***La planificación de la iteración que se divide en:***

        - **Selección de los requisitos del proyecto (4 horas máximo)**

            Aquí es donde el cliente presenta al equipo la lista de requisitos, se aclaran dudas al respecto, se seleccionan los que estarán listos para la próxima iteración y se les da una prioridad.

[3] Consultado de **http://www.proyectosagiles.org/que-es-scrum**

- **Planeación de la iteración (4 horas máximo)**

  Se definen las actividades y pasos a seguir para cumplir con la lista de objetivos y requisitos previamente definidos en conjunto con el usuario final.

b. Ejecución de la iteración

Se debe realizar una reunión de sincronización diaria (15 minutos máximo). En estas reuniones se evalúan las tareas que cada miembro del equipo está realizando y se ven aspectos como las dependencias entre tareas, el avance y si ha habido problemas u obstáculos a la hora de llevar a cabo dichas tareas, esto con el fin de realizar ajustes y poder cumplir con las fechas y entregables.

En estas reuniones cada miembro del equipo debe responder las siguientes preguntas:

- ✓ ¿Qué hice ayer?
- ✓ ¿Qué problemas he tenido?
- ✓ ¿Qué voy a hacer hoy?

c. Inspección y adaptación

Se debe llevar a cabo una reunión al final de cada iteración con el fin de evaluarla. Dicha reunión se divide en 2 partes:

1. ***Demostración (máximo 4 horas):***

   Se presentan al cliente los nuevos requisitos desarrollados. En este punto se pueden planear modificaciones de acuerdo con lo que el cliente requiera.

2. ***Retrospectiva (máximo 4 horas):***

   Se analiza y evalúa la forma en que el equipo ha venido trabajando.
   Se ven aspectos como los obstáculos, en qué se puede mejorar y qué han hecho muy bien.
   Esto sirve para buscar la mejora continua del equipo de trabajo con el fin de ser más productivos cada vez que se evalúan.

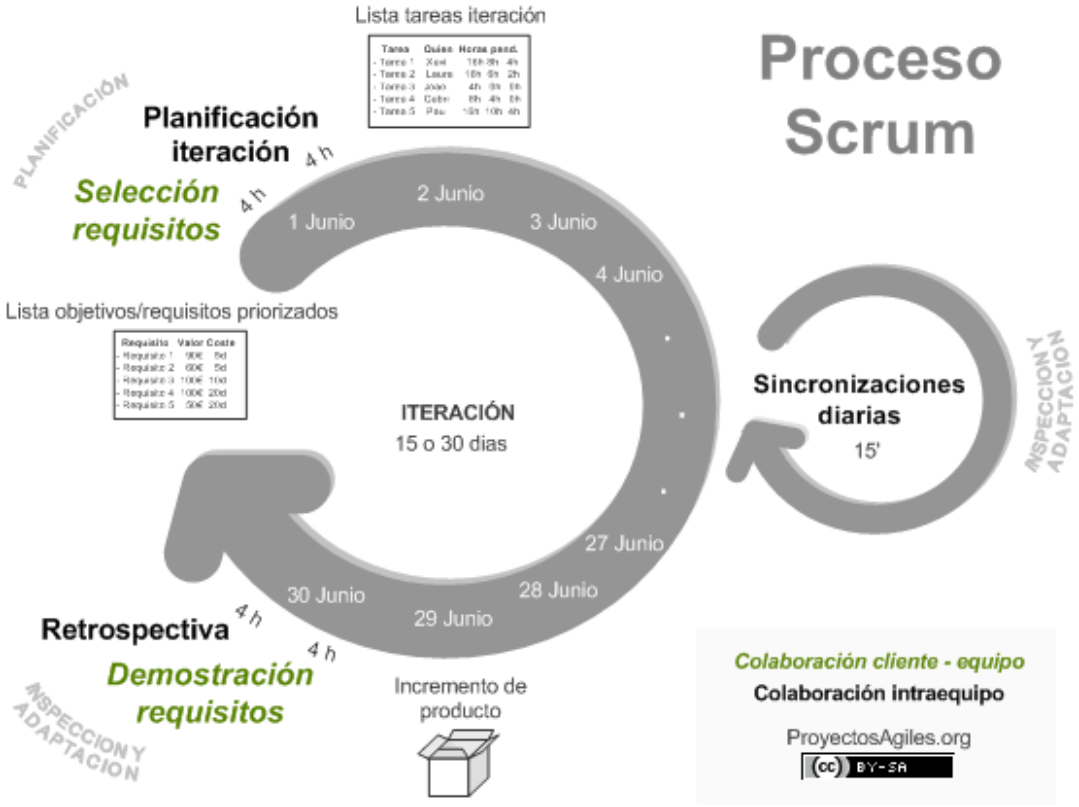

Figura 3.
Modelo *Scrum*
Tomado de http://www.proyectosagiles.org/que-es-scrum

Lean Software Development

Esta es una metodología dirigida a la administración o gestión de proyectos y no a los procesos de desarrollo de software como tal. Por eso se dice que es una metodología que complementa otras metodologías como XP o *Scrum*. No es excluyente, es decir, que el uso de esta no excluye las demás.

A la hora de hablar de *Lean* se pueden mencionar los siguientes principios:

1. Se elimina todo lo que para el cliente no tiene valor como código innecesario, atrasos en el proceso de desarrollo de software, burocracia, comunicación lenta o requisitos poco claros.

   Esto le quita complejidad al software y al proceso como tal. Muchas veces código no funcional o implementaciones parciales hacen al software complejo de entender y puede llevar a la generación de pulgas inesperadas.

2. Equipo de desarrollo y clientes deben aprender en conjunto. Las reuniones continuas y la presentación de avances en períodos cortos facilita esta tarea.

   La difusión del conocimiento entre cada miembro del equipo de desarrollo y clientes busca la solución de pulgas y la mejora continua de procesos.

3. Ver las soluciones que se proponen como opciones hasta no tener claros lo procesos, requerimientos y necesidades del cliente. Esto da flexibilidad al proyecto ya que se deja abierto a cambios y ajustes en el tiempo. Este es un punto que se debe tener muy claro cuando se trabaja con esta metodología.

   El software como producto por sí solo es considerado "abstracto" y más aún si es complejo. Cuanto más complejo sea un proyecto, más abierto a cambios, pues generalmente es más probable no tener claro qué quiere obtener.

   La solución del proyecto se considerará como tal cuando ya se tengan claramente definidos los objetivos, las necesidades y los requerimientos del software. Esto generalmente se obtiene en etapas avanzadas del proyecto.

4. Entregar el producto final lo más rápido posible y sin defectos considerables.

5. El equipo de desarrollo debe tomar las decisiones de implementación. Dar poder de decisión al equipo de trabajo.

6. El cliente debe tener un conocimiento general del sistema. Debe estar involucrado y conocer integralmente la solución.

7. Ver los diferentes componentes de un sistema como un todo. Nunca se debe perder la percepción de eso. Aunque cada componente se puede ver como un "sub proyecto" debemos tener claro que es una pieza más para el buen funcionamiento del sistema, de la solución integral y, por ende, debe funcionar acorde con las especificaciones y necesidades del cliente.

Feature Driven Development (FDD)

Esta es una metodología de software iterativa e incremental. Además, uno de los puntos fundamentales es ser manejada desde una perspectiva funcional del cliente. Esta pensado para proyectos de desarrollo de cortos.

Los siguientes son los principios a considerar o tomar en cuenta a la hora de implementar FDD.

1. Basado en un proceso iterativo en períodos cortos.

2. Las iteraciones se definen basadas en las funcionalidades, especificadas y definidas por el cliente.

3. Solo se considera para las fases de diseño y construcción del ciclo de vida de desarrollo de software.

4. Se debe considerar complemento de otras metodologías.

5. Se deben definir entregas tangibles del producto y realizar evaluaciones del proceso.

6. FFD define 3 categorías o grupos de roles:

   - ***Roles clave***

     a. Gerente de proyecto
     b. Arquitecto jefe.
     c. Gerente de desarrollo.
     d. Programador jefe.
     e. Propietarios de clases.
     f. Experto de dominio.

   - ***Roles de soporte***

     a. Administrador de entrega.
     b. "*Gurú del lenguaje*".
     c. Herramientista.
     d. Administrador del sistema.

   - ***Roles adicionales***

     a. Tester
     b. Escritor de documentos técnicos.

7. Rápida adaptación a cambios en los requerimientos del proyecto.

8. Cada fase del proceso debe tener claramente definidas las entradas, las tareas, las pruebas y el resultado esperado.

9. FDD define 5 actividades claves:

   1. Diseño del modelo general

   El proyecto inicia con la definición de alcances a un alto nivel. El modelo será refinado a lo largo del proyecto.

2. ***Construir la lista de funcionalidades***

   En esta actividad se realiza un análisis de cada uno de los procesos que el cliente desea automatizar.
   La lista de funcionalidades se construye basada en el modelo general previamente definido.

3. ***Planear por funcionalidad***

   En esta actividad se construye el plan de desarrollo.

4. ***Diseñar por funcionalidad***

   Se debe crear un paquete de diseño por cada funcionalidad definida.
   Se elige un grupo pequeño de funcionalidades que será desarrollado en un plazo de tiempo no mayor a las 2 semanas.
   Al final de esta se fase se realizará una evaluación del diseño creado por el equipo de trabajo. Una vez aprobado se refina el modelo general realizado en la primera fase.

5. ***Desarrollar por funcionalidad***

   En esta etapa se desarrollan las funcionalidades basadas en el análisis y diseño previamente realizados.
   Una vez desarrolladas e implementadas las funcionalidades deben ser sometidas a pruebas y revisión de código. Una vez aprobado, el código es agregado al *build* principal.

Rational Unified Process (RUP) [4]

RUP es una metodología de desarrollo de software que en conjunto con *UML* (Lenguaje Unificado de Modelado) constituye una de las metodologías que más se utilizan para el análisis, desarrollo y documentación de sistemas de información.

Su principal objetivo es obtener software de calidad dentro de un tiempo y presupuesto aceptables y previsibles.

A continuación, los principios que deben ser tomados en cuenta a la hora de implementar *RUP* en un equipo de trabajo.

1. *RUP* se basa en el avance del proyecto por medio de iteraciones debido a la complejidad de los sistemas modernos. Estas iteraciones nos ayudan a ir incrementando nuestro entendimiento del proyecto en el tiempo, además de darnos la opción de ir refinando nuestro proyecto, lo que nos da la posibilidad

[4] Consultado de **http://www.conexionit.com/blog/metodologias/que-es-rup.html**

de hacer cambios en los requerimientos del proyecto e irlo ajustando a las necesidades del usuario final o cliente.

2. El manejo de los requerimientos se realiza por medio de escenarios y casos de uso lo que ayuda a su definición, siendo estos la base de las fases de implementación o construcción y pruebas del sistema.

3. Se deben utilizar arquitecturas basadas en componentes y servicios.

4. Se deben modelar visualmente los procesos a implementar y automatizar. De esta manera es más fácil la comprensión de las tareas, actividades y objetivos del proyecto. *UML* es la base de *RUP* para la modelación visual.

5. Se deben de tomar en cuenta las pruebas para cada etapa del proyecto.

6. El ciclo de vida de *RUP* se divide en 4 etapas, de las cuáles cada una de ellas produce un nuevo producto. Dichas fases son las siguientes

    - ***Fase de inicio***

        En esta fase se define el alcance del proyecto, partes involucradas, se identifican los posibles escenarios y casos de uso, factores críticos de éxito, riesgos, estimación de recursos necesarios y el plan con las fechas importantes a lo largo del proyecto.

        De la fase inicio se pueden obtener los siguientes resultados:

        - Documento con visión general de los requerimientos a un nivel básico de las características y restricciones principales.
        - Modelo inicial de casos de uso (10% – 20% completados).
        - Glosario inicial del proyecto.
        - Pronóstico financiero.
        - Estimación de riesgo inicial.
        - Plan de proyecto con sus iteraciones definidas y especificadas.
        - Uno o más prototipos.

    - ***Fase de elaboración***

        El objetivo o propósito de esta fase es analizar el problema en cuestión y proponer una arquitectura adecuada y deseada para desarrollar el plan del proyecto. Es fundamental que para esta etapa se tenga una visión clara y completa del proyecto. Los objetivos y requerimientos del sistema a desarrollar deben ser claros y estar bien especificados y definidos.

Es importante recalcar que es al final de esta fase cuando ya se tiene definida la arquitectura a utilizar y lo requerido es cuando se define y se decide si se inicia con la codificación del sistema.

Para esta fase la arquitectura, los requerimientos y los planes deben ser bastante estables, y además, los riesgos minimizados. Es más fácil planear fechas y costos de una manera certera.

En esta fase se genera un prototipo de la arquitectura a utilizar en la siguiente fase. Este prototipo tratará los casos de uso más críticos del proyecto.

Al finalizar esta fase se deberán obtener los siguientes resultados:

- Modelos de casos de uso (80% completados).
- Requerimientos suplementarios.
- Descripción de la arquitectura de software a utilizar.
- Prototipo arquitectónico ejecutable.
- Plan de desarrollo para el proyecto total.

- ***Fase de construcción***

Esta es la fase de codificación y desarrollo del proyecto, basado en la arquitectura y plan propuestos en la fase anterior. En esta fase también se deben realizar todas las pruebas necesarias al software, con el fin de obtener un producto de calidad.

Como resultado de esta fase se obtiene un producto listo para poner en producción, además de los manuales de usuario y documento con la descripción de la versión o *release.*

- ***Fase de transición***

En esta fase se llevan a cabo todas las tareas o actividades necesarias, con el fin de hacer el pase del sistema desarrollado al ambiente de producción.

Como resultados se deben obtener:

- La prueba beta para validar las expectativas del usuario.
- Se debe realizar la operación del sistema nuevo en paralelo con el sistema anterior, esto con el fin de validar que todos los resultados y comportamientos del sistema nuevo sean los esperados.
- Entrenamientos y capacitación a los usuarios finales del sistema.

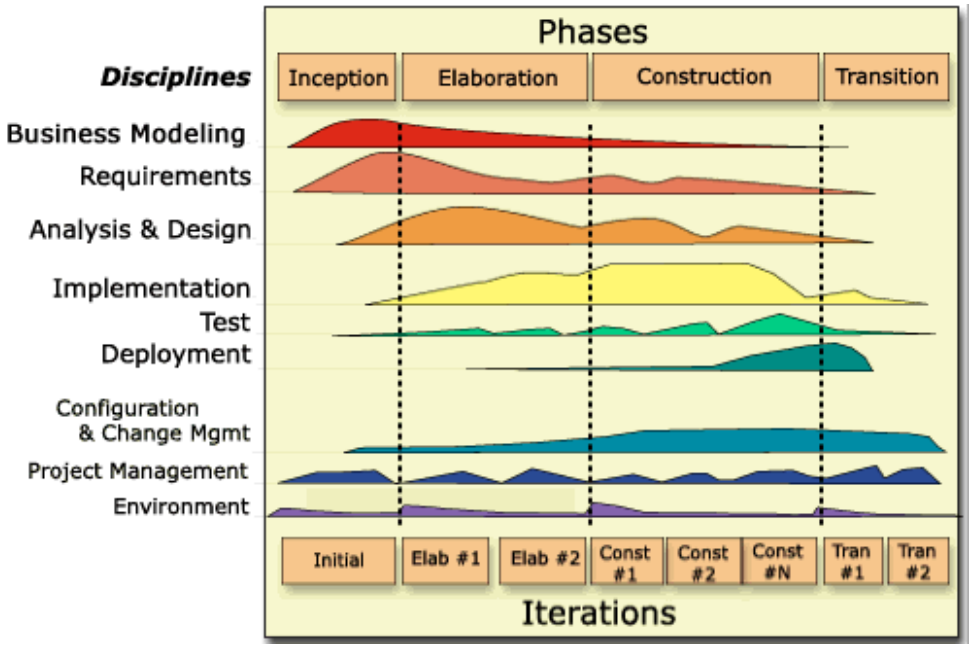

Figura 4.
Modelo *RUP*.
Tomada de http://www.conexionit.com/blog/wp-content/uploads/2008/10/rup12.gif

V-Model (Software Developement)

Se dice que el modelo V se basa o se deriva del modelo o metodología de desarrollo en cascada. Lo importante de este modelo y que vale la pena recalcar es que la fase de *testing* es primordial. En este modelo cada fase dentro del ciclo de vida del desarrollo de *software* tiene asociada su etapa respectiva de *testing.* Es decir, en forma paralela se va realizando la prueba durante el desarrollo de un sistema y es después de la fase de codificación donde se dobla hacia arriba para continuar con el *testing* y es ahí donde se forma la figura de la "V".

Esta metodología consta de dos fases, las que a su vez se subdividen en otras. El proceso de *V-Model* se describe a continuación:

1. ***Fase de verificación:***

Esta fase se compone de las siguientes etapas:

a. ***Análisis de requerimientos:***

Esta es la etapa donde se realiza un análisis de las necesidades y requerimientos del usuario. Una vez analizado el problema se define la lista de requerimientos. Esta lista de requerimientos es la base del proyecto y sistema a desarrollar. Producto de este análisis se obtiene el documento de requerimientos del usuario.

El documento de requerimientos del usuario generalmente contiene lo siguiente:

- Descripción de la funcionalidad del sistema.
- Descripción de la interface.
- Descripción del rendimiento.
- Descripción de la información o datos a procesar y almacenar.
- Descripción de la seguridad que el sistema debe implementar.

En esta etapa es muy importante la participación del usuario final ya que dependiendo de su perspectiva y de sus necesidades se define el documento de requerimientos.

La etapa de análisis de requerimientos está sujeta a evaluación o *testing.* La etapa de *testing* que le corresponde es la fase de ***Testing* de aceptación de usuario.** El documento es sujeto a evaluación y aprobación por parte del usuario final.

b. ***Diseño del sistema***

En esta fase los ingenieros de sistemas analizan y entienden el problema en cuestión, tomándose como base el documento de requerimientos del usuario definido en la etapa previa, con el fin de proponer soluciones o formas para llevar a cabo lo que el usuario necesita. Si por alguna razón existiera algún impedimento para desarrollar algún requerimiento, se le informa inmediatamente del problema al usuario. En este momento se analiza de nuevo el requerimiento, se ajusta y se modifica el documento de requerimientos con aprobación del usuario final.

El documento de especificaciones del software es creado en esta etapa. Este documento contiene información técnica de cómo se llevará a cabo la solución. Se describe información como:

- Organización del sistema.
- Estructura de datos a utilizar.
- Estructura de los menús.
- Escenarios de negocio.
- Ventanas a utilizar.
- Tipos de reportes a generar.
- Diagramas de entidades.
- Diccionario de datos.
- También se crean los documentos para la fase de *testing* del sistema.

c. Diseño de la arquitectura:

En esta etapa se define la arquitectura del software a un nivel macro. Dicha arquitectura consiste básicamente en definir la siguiente información:

- Lista de módulos a implementar y una descripción breve de la funcionalidad de cada uno de ellos, sus interfaces y relaciones.
- Dependencias.
- Tablas de bases de datos.
- Diagramas de arquitectura.
- Los elementos tecnológicos necesarios que ayudan a definir la arquitectura de cómo se implementará la solución de *software* propuesta.

En esta etapa se llevan a cabo las pruebas de integración.

d. Diseño del módulo:

En esta etapa cada módulo definido anteriormente se subdivide en otros módulos o unidades más pequeñas, donde cada de uno de ellos es analizado, con el fin de definir exactamente su funcionalidad y como van a interactuar entre ellos.

De esta manera se puede definir y dividir el trabajo en tareas y asignarlas a los desarrolladores para que empiecen a codificar.

Producto de ese análisis y subdivisión de los módulos es que se obtiene el documento de especificación del programa a un bajo nivel. Este documento contiene la siguiente información:

- Funcionalidad lógica detallada de cada módulo y submódulo identificado descrita en pseudocódigo.

- Descripción de las tablas de la base de datos con sus tipos de datos y tamaños.
- Definición detallada de las interfaces (referencias API) a utilizar.
- Dependencias.
- Lista de mensajes de error.
- Datos de entrada y de salida del sistema.

El diseño del *unit testing* es llevado a cabo en esta etapa.

2. Fase de validación:

a. Unit testing:

El *unit testing* es el primer paso del proceso de *testing* dinámico. En esta fase los desarrolladores de software escriben código con el fin de simular ciertos escenarios, en donde se definen parámetros o datos de entrada además de los resultados esperados. De esta manera se prueba que la funcionalidad del sistema y los diferentes escenarios arrojan los resultados esperados.

b. Pruebas de Integridad:

En esta fase todos los módulos separados se prueban en conjunto donde se prueba su comportamiento, funcionalidad e integridad entre los diferentes componentes con los cuales se definió su interacción.

c. Pruebas del sistema:

En esta fase se comparan las especificaciones del sistema contra el sistema actual. Las pruebas del sistema se llevan a cabo basadas en el documento de diseño del sistema creado en las fases previas de esta metodología.

Generalmente, las compañías utilizan herramientas de *software* desarrolladas con el fin de automatizar estas pruebas. De esta manera se asegura la "corrida" adecuada y en un menor tiempo la ejecución de las pruebas del sistema. Es muy importante que el equipo de *QA* confirme dichas pruebas.

d. Pruebas de aceptación de usuario:

Esta etapa es necesaria para documentar la aprobación del análisis de requerimientos y del sistema como tal por parte del usuario final.

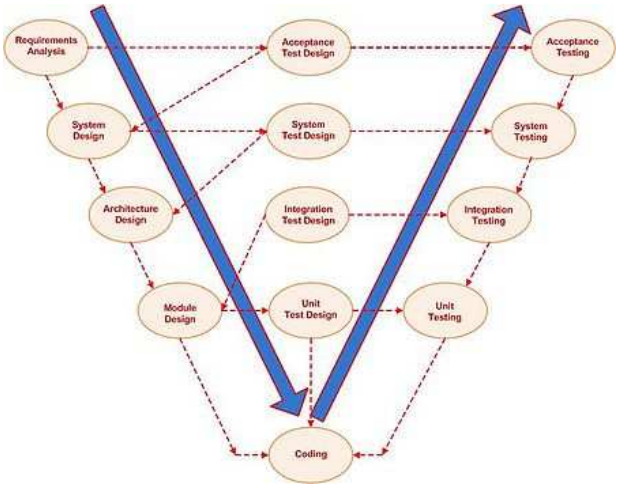

Figura 5.
Etapas de *V-Model*
Tomada de http://en.wikipedia.org/wiki/V-Model_(software_development)

Rapid Application Development (RAD)

Esta metodología se enfoca más en la codificación que en la planeación de las actividades y etapas del proyecto ya que uno de los aspectos importantes de *RAD* es iniciar con el desarrollo del sistema lo más pronto posible, con el fin de obtener un prototipo de lo que el usuario final está solicitando o requiere.

A diferencia del ciclo de vida tradicional, *RAD* comprime las fases de análisis, diseño, desarrollo y *testing* en una serie de iteraciones cortas del proceso de ciclo de vida de desarrollo de *software.*

De tal manera, por medio de estas iteraciones durante el desarrollo del proyecto, los desarrolladores y el usuario final de la aplicación van detallando y modificando el sistema hasta obtener el producto deseado.

Además, por ser el tiempo un factor tan crítico, en esta metodología es necesario o recomendado el uso de herramientas *CASE* que agilicen el proceso de desarrollo durante el proyecto.

En todos los proyectos el tiempo es un factor crítico, pero *RAD* especialmente tiene como uno de sus principales objetivos obtener un producto funcional que:

- Satisfaga las expectativas del usuario en el menor tiempo posible.
- A un bajo costo.
- De alta calidad.

RAD se enfoca primordialmente en los siguientes cuatro componentes:

1. ***Herramientas:***

   La tecnología hoy brinda muchas opciones y facilidades para llevar a cabo y desarrollar los proyectos de *software*. Actualmente, se tiene a disposición un sin número de herramientas que dependiendo de las dimensiones y alcance del proyecto se pueden tomar en cuenta para dicho trabajo.

   Algunas de estas herramientas son las siguientes:

   - Generadores de reportes.
   - Lenguajes de cuarta generación.
   - Herramientas de bases de datos.
   - Herramientas *CASE.*

   Sin duda alguna y, por la razón de ser de esta metodología, es de suma importancia la buena elección de este tipo de herramientas, ya que sin duda alguna la elección de ellas impactará en el desarrollo y tiempo en que se lleve a cabo el proyecto.

2. ***Gente:***

   Se incluyen el equipo de desarrolladores y los usuarios finales del sistema. *RAD* incluye al usuario final como parte del equipo de trabajo del proyecto, pues es necesaria su participación a lo largo del desarrollo del sistema para ir afinando y detallando los aspectos o requerimientos que en un inicio no quedaron claros o bien definidos. Al final de cada iteración se irá obteniendo cada vez un producto más completo y acorde con las necesidades del usuario final.

3. ***Metodología:***

   Como se mencionó anteriormente, la metodología se basa en el desarrollo de prototipos de manera conjunta con el usuario.

4. ***Administración:***

   Debe existir un compromiso por parte de la gerencia o administración del proyecto. Antes del inicio del desarrollo de la aplicación se deben tener definidos formalmente los procedimientos, documentos y entregables durante el proyecto. Otro aspecto importante es contar con los estándares a seguir para el proyecto.

Aspectos como:

- Recursos de datos.
- Aplicaciones.
- Sistemas.
- Plataformas de hardware.

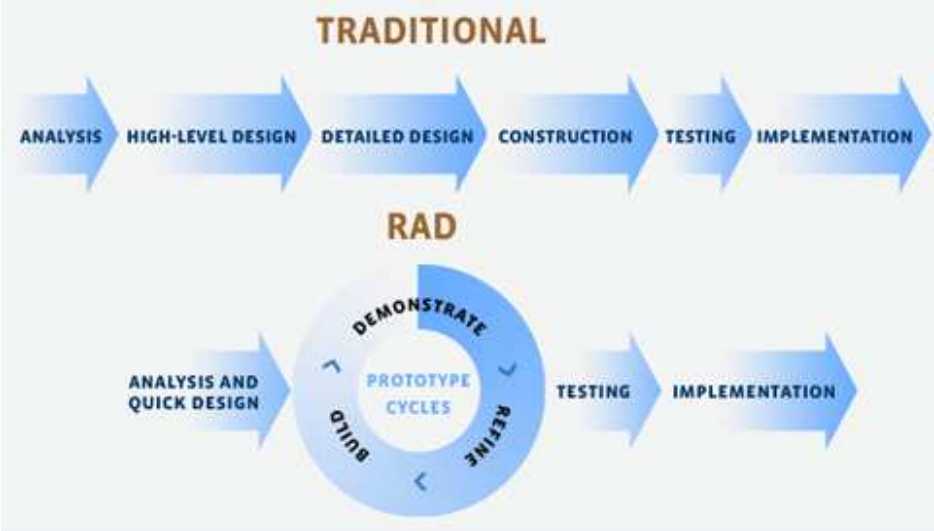

Figura 6.
Modelo *RAD* contra Modelo Tradicional
Tomado de http://www.etondigital.com/wp-content/post-images/rad-diagram.gif

Test Driven Development (TDD)

En esta metodología lo importante son las pruebas. Entonces, se propone básicamente enfocarse primero en las pruebas, con el fin de producir código limpio, eficiente y sin errores. Estas pruebas las escribe y crea el desarrollador del sistema antes de iniciar con el desarrollo de la aplicación (*Test First Development*) y luego por medio de refactorización se va escribiendo y mejorando el código del sistema.

La automatización de estas pruebas son los *Unit test* o pruebas unitarias. Estas pruebas deben ser diseñadas y escritas, basadas en los requerimientos del sistema. De esta manera, si cuando se ejecuten las pruebas se obtiene un resultado positivo (pasó la prueba) se está cumpliendo con los objetivos y requisitos. Si por el contrario, se obtiene un estado fallido de la prueba se debe rediseñar o rescribir el código (refactorización).

Cuando se habla de *TDD* se puede hablar de 3 principios básicos:

1. No se puede escribir código productivo si no es con el fin de pasar un test o prueba fallida.

2. No se puede escribir más de lo necesario para que falle un test. Los errores de compilación se consideran fallos.

3. No escribir más código productivo del necesario para pasar un test.

Estos principios o reglas nos sugieren mantener el código lo más simple posible, además de buscar hacerlo fácil de mantener y desarrollar código seguro a la vez.

El ciclo de vida de esta metodología se puede dividir en las siguientes etapas:

1. ***Se escribe el test unitario o unit test***

   En estos momentos, al ejecutar el test va a fallar pues no tenemos código implementado, solo la prueba.

2. ***Escribir o crear el código con el fin de pasar la prueba***

   En este punto del ciclo de vida se escribe el código funcional de la aplicación que será evaluado por la prueba unitaria o *unit test*.

   Lo que se busca en esta etapa es obtener el estado de que "pasó" la prueba.

   El código escrito debe llevarse a cabo respetando uno de los principios descrito previamente. Mantener el código lo más simple posible.

3. ***Ejecutar las pruebas***

   Producto de la ejecución, cada prueba tendrá su estado correspondiente asociado (Pasado o Fallido). Se deben elegir los tests que fallaron y se analizará de nuevo el código para posteriormente volver a correr las pruebas.

4. ***Refactorización***

   Este es el proceso de analizar y reescribir el código, donde el objetivo principal es que el test fallido pase.

   Esta etapa se realiza tantas veces como sea requerido hasta lograr el objetivo de que la prueba pase.

   También, como producto del análisis, puede ser necesario dividir la prueba unitaria en otras y crear nuevas pruebas.

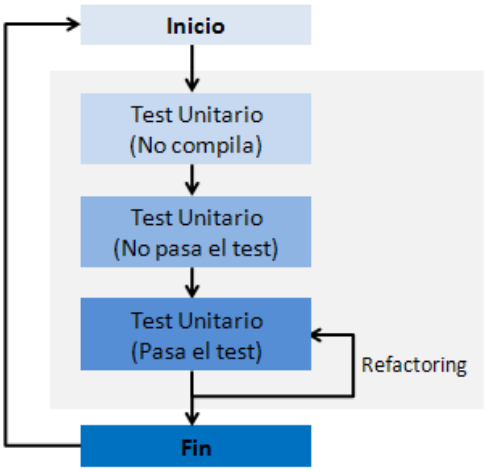

Figura 7
Modelo *Test Driven Development.*
Tomada de http://grimpidev.wordpress.com/category/test-driven-development/

Algunas de las ventajas que se pueden observar al implementar esta metodología son:

Para realizar las pruebas primero es muy importante contar con un panorama claro de lo que se pretende obtener, de cuáles son los requerimientos del sistema o las necesidades del usuario final. Esto nos garantiza iniciar con una visión clara de lo que se quiere. Muchos proyectos de software inician sin tener esta visión y generalmente influye de manera negativa en el proyecto.

Escribir código de la mano con la ejecución de pruebas unitarias produce un código más limpio.

Al tener un código más limpio, muchas veces se acorta el tiempo de desarrollo del proyecto, ya que al tener menos “pulgas” no se dedica tiempo a resolverlas.

Producir un código simple nos da como resultado un código fácil de entender y mantener, por lo que es más fácil la detección de “pulgas” o errores en el sistema.

Otra ventaja palpable es que si ingresa un desarrollador, la curva de aprendizaje es menor ya que se cuenta con código simple.

En la actualidad muchas herramientas de desarrollo de software ya cuentan con la opción de crear estas pruebas unitarias, de esta forma es más fácil la integración de este tipo de pruebas al código del proyecto.

Al implementar *Test Driven Development*, es fundamental realizar las pruebas al inicio del desarrollo del sistema, pues si no se hace de esta manera no se está implementando *TDD*.

Conclusiones y recomendaciones

Como ya se ha mencionado, el a*gile manifesto* nos da una serie de pasos o puntos que se deben cumplir. No dicen cómo llevarlos a cabo y es ahí donde surgen una serie de metodologías que nos sugieren cómo cumplir con los principios del a*gile manifesto.*

Uno de los puntos claros es que *agile* no es cumplir o hacer lo que dicen las metodologías estudiadas a lo largo de este artículo. Las metodologías son formas o pasos que nos sugieren cómo cumplir con los principios ágiles. Por eso es bueno hacer hincapié en que el cómo cumplir con los principios no es lo más importante, siempre y cuando el resultado sea lo que dicta el *agile manifesto.*

En estos días, los climas organizacionales, naturaleza del negocio y situaciones de las empresas son muy diferentes y es difícil adaptar ciertas metodologías al trabajo diario y proyectos de cada una de ellas. Por esta razón, el *agile manifesto* no dice cómo cumplir con sus principios. De esta forma tenemos las opciones de mezclar las mejores ideas, sugerencias o pasos de cada una de las metodologías y ajustarlas para implementarlas de la mejor manera en nuestras compañías o proyectos.

Otros aspectos a recalcar y presentes en muchas de las metodologías son los siguientes:

- La comunicación constante.
- Iteraciones.
- El rol del usuario final en la aplicación.
- Transferencia de conocimiento.

La comunicación constante

Trabajar como un equipo es de suma importancia y para lograr eso debemos expresar siempre nuestras ideas y comprender claramente los objetivos y tiempos del proyecto.

Por esta razón, debe existir una buena comunicación entre todos los miembros del equipo de trabajo con el fin de avanzar lo más rápido posible y solucionar los problemas que se puedan presentar de una manera oportuna, eficaz y eficiente.

Hablando de la comunicación podemos notar que muchas de las metodologías coinciden en este punto y sugieren la calendarización diaria o periódica de ciertas reuniones durante el proyecto. Durante estas reuniones se tocan aspectos como los problemas (si es que existe alguno), tareas actuales y pendientes. De esta

manera se conoce el estado del proyecto y se pueden tomar acciones fundamentales para ayudar al éxito del proyecto.

Iteraciones

El uso de iteraciones nos ayuda a tener control sobre lo que sucede en el proyecto. El control es muy importante pues nos ayuda a tener una visión clara de hacia adonde va el equipo y, consecuentemente, el proyecto.

Durante estas iteraciones se pueden ver aspectos como:

- Estado de las tareas actuales.
- Tareas pendientes.
- Revisiones de código.
- Ayuda o sugerencias para solucionar los problemas que se presenten.
- Colaboración entre los mismos miembros del equipo de trabajo.
- Reasignación de tareas, entre otros.

El rol del usuario final

Muchas metodologías apuntan o mencionan que el papel del usuario final es fundamental. Se debe incluir siempre al usuario final como miembro activo del equipo de trabajo. Miembro activo porque él es quien tiene más claro los procesos, requerimientos y objetivos de un determinado proceso, por ejemplo.

Entonces, se debe involucrar completamente al usuario final en todas las fases del ciclo de vida del proyecto. De esta manera, por medio de su evaluación podemos obtener un sistema o aplicación que cumpla a cabalidad con los requerimientos o necesidades existentes.

Transferencia de conocimiento

La transferencia de conocimiento ayuda mucho a que el proyecto avance como se espera. Es más fácil cumplir con los objetivos propuestos si todos los miembros del equipo de trabajo comprenden y entienden el estado del proyecto (adonde están), cómo se ha logrado hasta el momento lo que se tiene desarrollado y hacia adonde se pretende llegar.

Es posible mover de rol a un miembro del equipo, reasignar tareas y esperar colaboración de cada miembro pues todos conocen de todo.

Por ejemplo, una de las metodologías que se abarcaron en este artículo hace hincapié en este aspecto. Dicha metodología es *XP* o *eXtreme Programming,* y

una de las formas de llevar a cabo esto es por medio de *pair programming* o programación en parejas. Como ya se mencionó, dos programadores desarrollan y trabajan en una misma computadora al mismo tiempo. De esta manera, el conocimiento y experiencia de cada uno de los desarrolladores se aprovecha y se expande el conocimiento sobre la tecnología utilizada y la aplicación al mismo tiempo.

Los datos anteriormente descritos abarcan, en términos generales, los puntos más importantes a considerar a la hora de implementar las metodologías ágiles en nuestros proyectos u organizaciones.

Podemos utilizar todas estas sugerencias, consejos y metodologías y tomar lo que mejor nos convenga de acuerdo con nuestras necesidades empresariales y condición del proyecto o negocio. Podemos crear una mezcla de principios y metodologías y crear una propia, siempre y cuando cumplamos con los principios descritos en el *agile manifesto.*

No necesariamente debemos seguir y tomar lo que nos sugieren en las diferentes metodologías disponibles actualmente. Lo importante como ya se mencionó es cumplir con los principios ágiles.

Resumen ejecutivo

El presente trabajo tiene como objetivo el estudio y análisis del *agile manifesto.* Se estudian aspectos como qué es *agile manifesto,* cuáles son sus principios y cómo podemos implementarlo en nuestras compañías u organizaciones, donde el desarrollo de software es una de las tareas principales o fundamentales.

Hablar de *agile manifesto* es hablar de puntos u objetivos que idealmente debemos lograr, cumplir o tener en nuestras organizaciones. El cómo hacerlo o cómo cumplir con *agile* es otra meta por alcanzar y es ahí donde surgen las metodologías ágiles que vienen a sugerir cómo implementar el *agile manifesto* en nuestras empresas de desarrollo de software o unidades de tecnología, donde de igual manera se desarrollan aplicaciones o sistemas de información. Durante este trabajo se tocarán las metodologías más importantes a la hora de hablar de *agile.*

Existen muchas formas o metodologías de cómo implementar *agile,* pero depende de muchas variables o factores que se deben tomar en cuenta a la hora de decidir su implementación pues por sí sola una organización es muy diferente de otra y no se puede implementar alguna de las metodologías mencionadas en este trabajo de la misma manera en dos organizaciones diferentes.

El juicio, el sentido común y la experiencia son muy importantes cuando se trata de implementar éstas metodologías en nuestras compañías. Contar con procesos y proyectos ágiles es una tarea ardua y organizada.

Contar con procesos y proyectos ágiles en nuestras empresas o unidades de desarrollo de sistemas es muy importante, pues contribuyen a mejorar los procesos para que sean más efectivos y precisos. Esto es de suma importancia ya que cuando hablamos de software hablamos de un producto intangible y es importante tener control sobre él. *Agile* nos orienta y nos guía en cómo y de qué manera podemos lograr eso.

Abstract

This present work has as an objective to study and analyze the agile manifesto. It explores aspects or questions such as what does agile say, what are their principles, and how can we implement it in our companies or organizations? where software development is a fundamental or major task.

Talk about agile manifesto, is to speak of points or goals that ideally we should achieve, carry or have in our organizations. How to do or how to comply with agile is another objective to be attained, and that's where the agile methodologies emerge and come to suggest how to implement agile manifesto in our software development companies or technology units, where similarly you develop applications or information systems. During this work we are going to touch the most important agile methodologies which we can consider at the time we want to implement this.

There are many ways about how to implement agile manifesto, but it depends on a several variables and factors we should take in mind at the time to decide to implement agile. Every organization is different, in this way we cannot pretend to implement some of the mentioned agile methodologies in this work in two companies in the same manner.

Judgment, common sense and experience are very important when it comes to implementing these methodologies in our companies. Achieving have agile processes and projects is a difficult task and organized.

Have agile processes and projects in our companies or development software units is very important because it helps us to improve processes, become more effective and accurate in our work. This is critical because when we talk about a software product is intangible and it’s important to have control over it. Agile orients us and guides us on how and how we can achieve that.

Frases descriptoras

- ✓ *agile manifesto*
- ✓ Metodologías ágiles
- ✓ ¿Cómo implementar *agile?*
- ✓ Implementación de metodologías ágiles
- ✓ Desarrollo de software ágil

Bibliografía

Agile Manifesto, 2001. *Manifesto for Agile Software Development.* Recuperado el 3 de octubre del 2009 de www.agilemanifesto.org

Agile Spain. Scrum. Recuperado el 24 de octubre del 2009 de http://www.agile-spain.com/donde_aprender_mas_sobre_scrum

Cleanroom Software Engineering. Recuperado el 3 de octubre del 2009 de http://en.wikipedia.org/wiki/Cleanroom_Software_Engineering

Desarrollo de software ágil. Recuperado el 3 de octubre del 2009 de http://en.wikipedia.org/wiki/Agile_software_development

Desarrollo de software ágil. Recuperado el 20 de octubre del 2009 de http://www.monografias.com/trabajos67/metodologia-desarrollo-softwares/metodologia-desarrollo-softwares2.shtml

Dynamic Systems Development Method. Recuperado el 9 de octubre del 2009 de http://en.wikipedia.org/wiki/Dynamic_Systems_Development_Method.

eXtreme Programmimg. Recuperado el 22 de octubre del 2009 de http://www.clubdevelopers.com/prog/articulos/xp/downloads/xp.pdf

Feature Driven Development. Recuperado el 31 de octubre del 2009 de http://en.wikipedia.org/wiki/Feature_Driven_Development

Introducción a *Test Driven Development.* Recuperado el 21 de noviembre del 2009 de http://grimpidev.wordpress.com/category/test-driven-development/

Lean Software Development. Recuperado el 30 de octubre del 2009 de http://www.qualitrain.com.mx/index2.php?option=com_content&do_pdf=1&id=159

Métodos Ágiles. Recuperado el 21 de octubre del 2009 de http://carlosreynoso.com.ar/archivos/carlos-reynoso-alternativas-metodologicas-en-arquitectura-de-software.ppt

Metodologías de desarrollo ágil. Recuperado el 30 de octubre del 2009 de http://www.qualitrain.com.mx/index2.php?option=com_content&do_pdf=1&id=159

Modelo Iterativo. Recuperado el 9 de octubre del 2009 de http://en.wikipedia.org/wiki/Iterative_and_incremental_development

Proyectos Ágiles. ¿Qué es Scrum? Recuperado el 24 de octubre del 2009 de http://www.proyectosagiles.org/que-es-scrum

Rapid Application Development. Recuperado el 18 de noviembre del 2009 de http://www.google.co.cr/imgres?imgurl=http://www.etondigital.com/wp-content/post-images/rad-diagram.gif&imgrefurl=http://www.etondigital.com/services/&h=273&w=480&sz=28&tbnid=cYuH0-VfEN8C2M:&tbnh=73&tbnw=129&prev=/images%3Fq%3DRapid%2BApplication%2BDevelopment&hl=es&usg=__gnwg04jRBi1nSugXl9DfAT9E05o=&ei=jywES-aLK82YlAfdkOTrAQ&sa=X&oi=image_result&resnum=4&ct=image&ved=0CBwQ9QEwAw

Rapid Application Development (RAD). Recuperado el 18 de noviembre del 2009 de http://www.hit.ac.il/staff/leonidM/information-systems/ch32.html

Rapid Application Development and User Centric Software Design as methodology. Recuperado el 18 de noviembre del 2009 de http://www.cirrussoftware.com/Sitemanager/userFiles/downloads/1_RADmethodology.PDF

SoftQA Network. V-Model. Recuperado el 14 de noviembre del 2009 de http://www.softqanetwork.com/2009/06/el-famosisimo-modelo-v/

Software Engineering Institute, Carnegie Mellon. Cleanroom Software engineering Reference. Recuperado el 4 de octubre del 2009 de http://www.sei.cmu.edu/library/abstracts/reports/96tr022.cfm

Test Driven Development. Recuperado el 19 de noviembre del 2009 de http://www.dosideas.com/wiki/Test_Driven_Development

Test Driven Development. ¿Por qué hacer pruebas antes? Recuperado el 19 de noviembre del 2009 de http://geeks.ms/blogs/lfraile/archive/2008/02/19/test-driven-development-pruebas-antes-o-despu-233-s.aspx

Tutorial IT Metodologías ¿Qué es RUP? Recuperado el 06 de noviembre del 2009 de http://www.conexionit.com/blog/metodologias/que-es-rup.html

V-Model (Software Development). Recuperado el 13 de noviembre del 2009 de http://en.wikipedia.org/wiki/V-Model_(software_development)