# DISEÑO E IMPLANTACIÓN DE UNA TECNOLOGÍA PARA EL DESARROLLO DE APLICACIONES ENFOCADAS EN LA UTILIZACIÓN DE LAS METODOLOGÍAS ÁGILES

MONOGRAFÍA

MARLON ALEXANDER PINEDA VÁSQUEZ

UNIVERSIDAD EAFIT
DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS
MEDELLÍN
2011

# DISEÑO E IMPLANTACIÓN DE UNA TECNOLOGÍA PARA EL DESARROLLO DE APLICACIONES ENFOCADAS EN LA UTILIZACIÓN DE LAS METODOLOGÍAS ÁGILES

MARLON ALEXANDER PINEDA VÁSQUEZ

Monografía para optar al título de profesional en Ingeniería de Sistemas

Asesor

JUAN PABLO RAMÍREZ
Ingeniero de Sistemas

UNIVERSIDAD EAFIT
DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS
MEDELLÍN
2011

Nota de aceptación

_____

_____

_____

_____

_____

_____

Presidente del jurado

_____

Jurado

_____

Jurado

Medellín 10 de Octubre de 2010

*“A todas aquellas personas que siempre estuvieron apoyándome de alguna u otra manera para que fuese posible este trabajo, a mi hermana que desde lejos me dio ánimo y consejos, a mis tías Celina y Emilse que siempre estuvieron presentes a lo largo de mi carrera apoyándome tanto económicamente como con sus consejos y sabiduría. A mis compañeros de estudio que siempre fueron un gran apoyo y me enseñaron mucho con su dedicación y disciplina. Al municipio de Medellín por su apoyo económico y por permitirme estudiar en esta gran universidad. Y al resto de mis amigos y compañeros de trabajo que me animaron, regañaron y exhortaron a seguir adelante con este trabajo y su realización”*

# AGRADECIMIENTOS

El autor expresa su agradecimiento a:

Juan Pablo Ramírez, mi asesor por su gran paciencia y ayuda para la elección del tema y su desarrollo, por su dedicación y pro actividad en pos de lograr sacar adelante este proyecto.

Camilo Herrera por su ayuda en las correcciones gramáticas y sintácticas.

# ÍNDICE GENERAL

# ÍNDICE DE TABLAS

# ÍNDICE DE FIGURAS

# INTRODUCCIÓN

La ingeniería de software y el desarrollo de software son actividades que se deben ejecutar de manera conjunta para obtener resultados deseados y lograr satisfacer las necesidades de un contexto determinado para una solución empresarial o de otro ámbito. Actualmente, existen muchos enfoques y metodologías para ejecutar el desarrollo de software y el proceso que este conlleva. Dichos enfoques se han dividido en 2 grandes ramas: metodologías ágiles y metodologías tradicionales. Cada una de ellas tiene sus ventajas y desventajas de acuerdo a los tipos de proyecto que se ejecuten, y pueden ser convenientes o recomendables para cierto tipo de problemas mientras que para otros no.

Enfocándose en los tipos de proyecto Web a los cuales se le puede aplicar de manera exitosa una metodología ágil, aún existen varios inconvenientes a la hora de implementarse el proceso para el desarrollo de software, puesto que aunque la promesa del desarrollo ágil es de velocidad en la creación de software, no existe una herramienta que por sí sola nos permita sacar ventaja de las virtudes de tal enfoque.

Las etapas de análisis y diseño de sistemas juegan un papel de suma importancia para el desarrollo de aplicaciones robustas, completas y bien estructuradas, pero no son las de mayor peso al concebir un sistema, puesto que el desarrollo ocupa un 70 % del trabajo que se debe realizar para llegar al producto final. A la hora de implementar un diseño de software para la Web, muchas empresas y/o personas no cuentan con las herramientas y metodologías adecuadas para llevar a cabo dicha labor, debido a que una aplicación cuenta con múltiples capas y cada una de ellas puede llegar a necesitar de herramientas distintas e incluso diferentes lenguajes de programación. Aplicando una metodología ágil a proyectos de desarrollo en Java, el programador requiere gran pericia a la hora de escoger las herramientas apropiadas y más aún, integrarlas, puesto que no existe una suite que permita llevar un proyecto desde su

diseño a la creación pasando por la arquitectura, almacenamiento en base de datos, lógica de negocio y vista al usuario.

Por todas estas razones y por falta de una herramienta de integración y facilidad para el desarrollo de aplicaciones en Java que integre las actividades y ventajas de una metodología ágil se pretende sugerir y proponer una tecnología que permita crear software de forma rápida y sin muchas complicaciones.

# 1. OBJETIVO GENERAL

Diseñar y elaborar una metodología basada en las mejores prácticas de las metodologías ágiles utilizadas en el medio actual, que permita construir de forma rápida y eficiente cualquier aplicación en proyectos de pequeña y mediana envergadura.

## 1.1. OBJETIVOS ESPECÍFICOS

- Proponer una tecnología que a partir de algunas herramientas open source, y simplificando e implementando algunas de las practicas del enfoque del desarrollo ágil genere aplicaciones en Java de forma rápida y de acuerdo a las necesidades actuales de la región.

- Generar un estado del arte y un análisis de escenarios de uso de una metodología ágil transversal propuesta en base a las estudiadas de la cual la tecnología propuesta implementará.

- Realizar una exploración rápida de varias metodologías ágiles, sus referentes en casos de éxito, características, ventajas e inconvenientes, a fin de elegir los puntos más adecuados para este proyecto teniendo en cuenta los escenarios actuales y factores como: tiempo, dinero, curvas de aprendizaje, curva de apropiación de lógica de negocio y comunicación del equipo de desarrollo

# 2. ALCANCE

La tecnología a desarrollar podrá ser adaptada utilizando algunas herramientas open source recomendadas para Java. No se explicará su uso detallado pero sí en qué actividades y para qué artefactos se pueden utilizar. Se planteará una metodología alterna transversal de acuerdo a la experiencia regional y teniendo en cuenta las variables principales: curva metodológica, tecnológica y de conocimiento del negocio

# 3. JUSTIFICACIÓN

## 3.1. BENEFICIARIOS

Este proyecto está orientado a satisfacer las necesidades de aquellos programadores y diseñadores de software que utilizan alguna metodología tradicional o no tienen una establecida, necesitan de una metodología ágil y carecen de un framework para el desarrollo de aplicaciones que corresponda con su entorno, forma de trabajo y necesitan de una metodología que agilice su labor. También está concebido para mejorar la velocidad con la que un desarrollador diseña y crea las aplicaciones que le son necesarias.

## 3.2. IMPORTANCIA DEL PROBLEMA DENTRO DE LA CARRERA

La importancia de este proyecto dentro de la carrera se ve reflejada en la utilización del conocimiento obtenido en las áreas de ingeniería de software y desarrollo de software. Además del empleo de las habilidades en investigación, metodologías para el desarrollo de software y gestión del proceso software.

La aplicación de la temática aquí propuesta puede ser usada tanto a nivel académico como empresarial y permitirá tener un punto de vista más humano y completo del proceso de desarrollo de software, el cual puede marcar la diferencia a la hora de crear aplicaciones, su tiempo de desarrollo y la calidad final del producto obtenido.

# 4. METODOLOGÍAS ÁGILES DE DESARROLLO DE SOFTWARE

Una metodología en el ámbito de desarrollo de software, es un marco de trabajo que define que procesos, actividades y artefactos se producen, cuál es el orden de los procesos, quién ejecuta las actividades y qué métricas, estándares o reglas deben seguirse.

Para tener una idea de dónde vienen y qué plantean las metodologías ágiles hay que referirse al punto de partida "El Manifiesto ágil", el cual fue establecido por varios críticos que estuvieron de acuerdo en cambiar algunos de los precedentes del desarrollo de software.

## 4.1. EL MANIFIESTO ÁGIL.

> Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:
>
> **Individuos e interacciones** sobre procesos y herramientas
> **Software funcionando** sobre documentación extensiva
> **Colaboración con el cliente** sobre negociación contractual
> **Respuesta ante el cambio** sobre seguir un plan
>
> Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda.[1]

## 4.2. PRINCIPIOS DEL MANIFIESTO ÁGIL.

Sintetizando, estos son los 12 principios que plantea el manifiesto:

[1] Manifesto for Agile Software Development. [Artículo en Internet]. *http://agilemanifesto.org/* [Consultado en Septiembre del 2011]

1. Satisfacer al cliente mediante entregas tempranas y continuas de software con valor
2. Aceptación de requisitos cambiantes incluso en etapas tardías
3. Entrega de software funcional frecuentemente, entre 2 semanas y 2 meses, usando el periodo de tiempo más corto posible.
4. Desarrolladores y responsables del negocio trabajan juntos durante todo el proyecto
5. Desarrollo de proyectos con individuos motivados. Proveerles un buen entorno, apoyo y confiarles la ejecución del trabajo.
6. Conversación cara a cara como medio efectivo y eficiente de comunicación.
7. Software funcionando como medida del progreso.
8. Promoción del desarrollo sostenible. Promotores, desarrolladores y usuarios deben mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejoran la agilidad
10. La simplicidad o el arte de maximizar la cantidad de trabajo no realizado es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto - organizados.
12. a intervalos regulares, el equipo reflexiona sobre ser más efectivos para ajustar y perfeccionar su comportamiento en consecuencia.

## 4.3. METODOLOGÍAS ÁGILES VS TRADICIONALES.

Las metodologías se han dividido en ágiles y tradicionales, teniendo en cuenta su grado de complejidad y otros factores que a continuación se presentan.

**Tabla 1. Metodologías ágiles vs tradicionales**[2]

| METODOLOGIAS ÁGILES | METODOLOGIAS TRADICIONALES |
|---|---|
| Basadas en heurísticas de prácticas de producción de código | Basada en normas de estándares seguidas por entornos de desarrollo |
| Especialmente preparados durante el proyecto | Cierta resistencia al cambio |
| Impuestas internamente (por equipos) | Impuestas externamente |
| Proceso menos controlado, con pocos principios | Proceso más controlado, numerosas políticas/normas |
| No contratos tradicionales o al menos bastante flexible | Contrato prefijado |
| Cliente es parte del equipo de desarrollo | Cliente interactúa mediante reuniones |
| Grupos pequeños, menor a 10 personas en el mismo sitio físico | Grupos grandes distribuidos |
| Pocos artefactos | Más artefactos |
| Pocos roles | Más roles |
| Menor énfasis en la arquitectura del software | La arquitectura es esencial, expresada mediante modelos |

[2] Tabla tomada de: *http://es.scribd.com/doc/55710897/Metodologias-Agiles.pdf* [Publicación en Internet][Consultado en Septiembre del 2011]

## 4.4. ALGUNAS METODOLOGÍAS ÁGILES.

### 4.4.1. Metodología XP (extreme programation).

#### Características

- Trabajo en equipo.
- Aprendizaje de desarrolladores.
- Buen ambiente de trabajo.
- Realimentación continua entre cliente - equipo de desarrollo.
- Comunicación fluida.
- Simplicidad de soluciones implementadas.
- Coraje para enfrentar los cambios
- Especial para proyectos con requisitos imprecisos y cambiantes con alto riesgo técnico

#### Roles

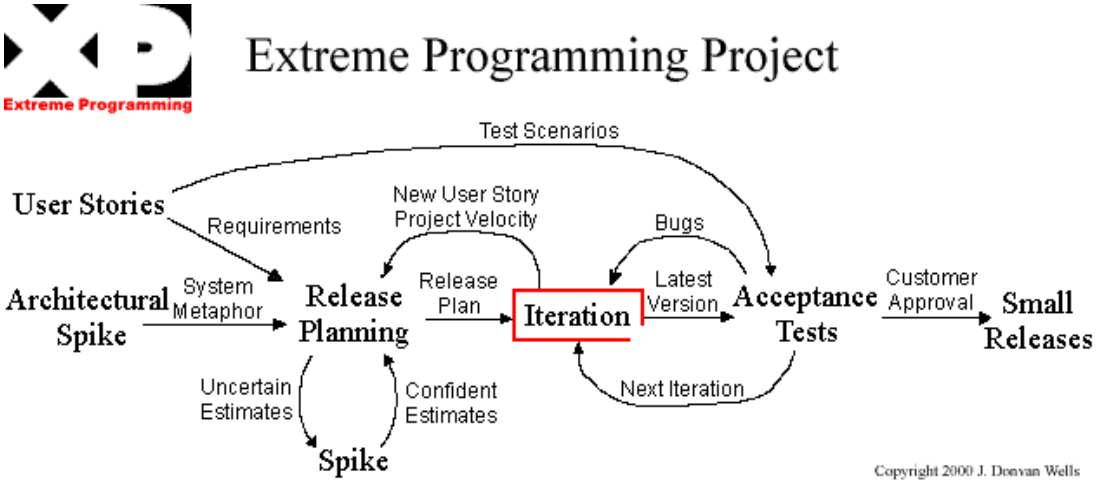
- Programador: Produce código y pruebas unitarias
- Cliente: Escribe Historias de usuario, participa de las pruebas funcionales, asigna prioridades a historias de usuario y elige cuales se implementan.
- Encargado de pruebas (tester): ayuda al cliente a ejecutar las pruebas funcionales, ejecuta pruebas regularmente, difunde resultados y se encarga de las herramientas de soporte a pruebas
- Encargado de Seguimiento (tracker): Hace realimentación al equipo, estimaciones de tiempo vs lo ejecutado y el seguimiento al proceso de cada iteración.
- Entrenador (Coach): Responsable del proceso global, proporciona las guías para hacer bien el proceso

- Consultor: Miembro externo con conocimiento específico en los temas, ayuda en el surgimiento de algún problema.
- Gestor (Big boss): Es el vínculo entre clientes y programadores, ejecuta la labor de coordinación.

## Proceso

1. El cliente define el valor del negocio a implementar.
2. El programador estima el esfuerzo para hacer la implementación.
3. El cliente selecciona lo que se va a construir (Prioridades y Restricciones de tiempo).
4. El programador construye.
5. Se repite el paso 1.

**Figura 1. Diagrama de Flujo de XP**[3]




[3] Imagen tomada de: Extreme Programming. [Artículo en Internet]. http://www.extremeprogramming.org/map/project.html [Consultado en Septiembre del 2011]

### Fases o ciclo de vida ideal

- Exploración
- Planificación de entrega
- Iteraciones
- Producción
- Mantenimiento
- Muerte del proyecto

### Prácticas

- **El juego de la planificación:** El equipo técnico realiza estimaciones de esfuerzo de implementación de las historias de usuario y los clientes deciden el ámbito, tiempo y entregas de cada iteración.
- **Entregas pequeñas:** Producir rápidamente versiones del sistema que sean operativas, una entrega no debería tardar más de 3 meses.
- **Metáfora:** metáfora es una historia compartida que describe como debería de funcionar el sistema (conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema ayudando como nomenclatura de clases y métodos del sistema).
- **Diseño simple:** Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.
- **Pruebas:** Producción de código dirigida por pruebas unitarias, son ejecutadas constantemente ante cada modificación del sistema.
- **Refactorización:** Actividad constante de reestructuración del código con el fin de remover duplicación de código, mejor legibilidad, simplificarlo y hacerlo más flexible a futuros cambios. se mejora la estructura interna del código sin alternar su comportamiento externo.
- **Programación en parejas:** Todo el código debe realizarse en pareja esto conlleva ventajas implícitas (menor tasa de errores, mejor diseño, mayor satisfacción de programadores).

- **Propiedad colectiva del código:** Cualquier programador puede cambiar cualquier parte del código en cualquier momento.
- **Integración continua:** Cada pieza del código es integrada en el sistema una vez que esté lista. El sistema puede ser integrado y construido varias veces en un mismo día.
- **40 horas x semana:** No se trabaja más de 40 horas por semana, no se trabajan horas extras en 2 semanas seguidas: si esto ocurre hay problemas que deben corregirse.
- **Cliente in-Situ:** El cliente debe estar presente y disponible todo el tiempo para el equipo de trabajo, la comunicación oral es más efectiva.
- **Estándares de programación:** Comunicación de los programadores es a través del código, seguir ciertos estándares para mantener legible el código.

### 4.4.2. Metodología Scrum

#### Características.

- Usado en entornos complejos
- Entrega de resultados de manera rápida
- Aceptación de requisitos cambiantes
- Innovación
- Competitividad
- Flexibilidad
- Productividad
- Proceso de desarrollo como “caja negra controlada”, no es proceso completamente definido
- El proceso no es lineal
- Entregas mensuales “Sprint” , mínimo cada 2 semanas
- Reuniones breves diarias para: exponer avances, problemas y posibles soluciones
- Al inicio de cada Sprint se establece una lista de requerimientos que no se pueden modificar llamada Backlog, que debe completarse cuando este finalice.

### Roles

- **Dueño del producto:** Responsable de la planificación, características del producto, priorización, fechas de lanzamiento y contenido, aceptar o rechazar resultados y asegurar la rentabilidad del producto.
- **Scrum Master:** Facilitador y líder del equipo que trabaja en contacto estrecho con el dueño del producto, debe estar informado de todo y es el que dirige al equipo.
- **Equipo:** Debe ser poli-funcional, compuesto por 7 miembros +- 2, su labor consiste en seleccionar el objetivo final de cada Sprint, especificar los resultados del trabajo y llevarlo a cabo. El equipo debe ser auto-organizado.

### Proceso

- Planificación de la iteración
    - Selección de requisitos (4 horas máximo)
    - Elaboración de lista de tareas de la iteración, estimación y asignación
- Ejecución de la iteración
    - Reunión diaria de sincronización
- Inspección y adaptación

  Al final de la iteración:
    - Demostración (4 horas máximo): se presenta lo hecho y se replanifica
    - Retrospectiva (4 horas máximo): Se analiza la forma de trabajo y se mejora.

**Figura 2. Proceso SCRUM**[4]


## Fases o ciclo de vida ideal

- Revisión de los planes de liberación de versión.
- Distribución
- Revisión y ajuste de los estándares de producto
- Sprint
- Revisión de Sprint
- Cierre

## Prácticas

- **El backlog de producto:** Es un repositorio de requerimientos (alto nivel) enunciados por los interesados en el éxito del proyecto
- **El backlog de versión (release backlog):** Consiste en una lista de requerimientos extraída del backlog de producto, priorizada para la próxima versión del producto
- **El backlog de sprint:** Se arma al principio de cada sprint, y reúne aquellos requerimientos que el equipo se compromete a completar para cuando finalice dicho sprint.

[4] Imagen tomada de: Anova Ingeniería de Software. [Publicación en Internet]. *http://www.anovanet.es/metodologia/metodologia.html* [Consultado en Septiembre del 2011]

- **Reuniones de Scrum:** Se llevan a cabo diariamente, aunque puede acordarse con el equipo para hacerla con una periodicidad diferente, es importante que se ejecute en el mismo lugar y a la misma hora y que su duración no sea mayor a 30 minutos. a cada miembro del equipo se le hacen 3 preguntas respecto a su última situación antes de la reunión: Qué hizo, qué obstaculizó su trabajo y qué planea hacer. Sobre las respuestas y sus problemas asociados pueden agendarse reuniones de subgrupo para darles solución.

### Controles Aplicados al Proceso Scrum

Son las herramientas imprescindibles para que el proyecto finalice a buen término. Son una serie de aspectos que se controlan y miden de forma tal de no anular la cualidad de "caja negra" que caracteriza las etapas de desarrollo y son:

- **Puntos del backlog:** Requerimientos funcionales del producto que no son cumplidos adecuadamente por la actual versión. Bugs, defectos, mejoras pedidas por el usuario, funcionalidad competitiva y actualizaciones tecnológicas son otros posibles puntos del backlog.
- **Versión (release):** Un conjunto de puntos del backlog que, en un determinado momento, representan una nueva versión viable del producto, sobre la base de las variables de requerimientos, tiempo, calidad y competencia.
- **Paquetes:** Componentes del producto u objetos que deben cambiarse para implementar un punto del backlog en una nueva versión del producto.
- **Cambios:** Cambios que deben ocurrir en un paquete para implementar un punto del backlog.
- **Problemas:** Problemas técnicos que suceden y deben resolverse para implementar un cambio.
- **Riesgos:** Los riesgos que afectan el éxito del proyecto son continuamente evaluados y se planean respuestas. Otros controles se ven afectados como consecuencia del análisis de riesgo.
- **Soluciones:** Soluciones a los problemas y riesgos, que habitualmente derivan en cambios.

- **Cuestiones (issues):** Cualquier otra cuestión que afecte al proyecto, y que no se defina en términos de paquetes, cambios o problemas.

### 4.4.3. Metodologías Crystal

#### Características

- Aspecto humano del equipo
- Tamaño reducido del equipo
- Comunicación dentro del equipo
- Políticas a seguir
- Espacio físico de trabajo, el mismo lugar de trabajo disminuye el costo de comunicarse.
- Combinación de productividad y tolerancia (mayor importancia al individuo)
- Reflexión sobre mejora de procesos
- Entregas Frecuentes
- Seguridad Personal (participación activa del equipo)
- Enfoque en las tareas individuales por parte del equipo
- Enfoque en la dirección del proyecto
- Acceso fácil a usuarios, testeo automático, manejo de la configuración e integración frecuente
- Expertos
- Ambiente técnico
- El desarrollo de software como un juego cooperativo
- Entregas (timebox) cada 2 semanas o cada mes
- Un timebox finaliza con código integrado y testeado

## Roles

- **Patrocinador Ejecutivo (Executive Sponsor ):** Establece los objetivos del proyecto con prioridades de compromiso. Consigue los recursos y define la totalidad del proyecto.
- **Coordinador (Coordinator ):** Con la ayuda del equipo, produce el mapa de proyecto, el plan de entrega, el estado del proyecto, la lista de riesgos, el plan y estado de iteración y la agenda de visualización.
- **Experto en el Dominio (Domain Expert):** Debe conocer las reglas y políticas del negocio.
- **Experto de uso (Usage Expert):** Junto con el experto en el dominio produce la lista de actores -objetivos y los archivos de casos de uso y requerimientos. Debe familiarizarse con el uso del sistema, sugerir atajos de teclado, modos de operación, información a visualizar simultáneamente, navegación.
- **Programador Diseñador (Designer-Programmer):** Produce, junto con el diseñador principal, los borradores de pantallas, el modelo común de dominio, las notas y diagramas de diseño, el código fuente, el código de migración, las pruebas y el sistema empaquetado.
- **Diseñador principal (Designer):** Tiene roles de coordinador, arquitecto, mentor y programador más experto
- **Tester:** Produce el reporte de bugs. Puede ser un programador en tiempo parcial, o un equipo de varias personas.
- **Escritor**. Produce el manual de usuario.

## Prácticas

- Periodo de 2 horas diarias donde el desarrollador no tendrá interrupciones
- Asignar un desarrollador a un proyecto por al menos 2 días antes de ser cambiado a otro proyecto

- **Exploración de 360 grados:** Se debe verificar o tomar una muestra del valor de negocio del proyecto, los requerimientos, el modelo de dominio, la tecnología, el plan del proyecto y el proceso.
- **Victoria temprana**. Pequeños triunfos iniciales sobre grandes triunfos tardíos
- **Re-arquitectura incremental**: No conviene interrumpir el desarrollo para corregir la arquitectura, la arquitectura debe evolucionar de tal manera que mantenga el sistema en ejecución mientras esta se modifica.
- **Radiadores de información:** Cartelera o lámina que se pone en algún lugar donde el equipo pueda observarla, tiene que ser entendible de forma rápida y renovada periódicamente.
- **Entrevistas de proyectos:** Se debe entrevistar a más de un responsable para tener visiones más completas.
- **Talleres de reflexión:** Dedicación de 30 a 60 minutos para reflexionar sobre: convenciones de trabajo, inconvenientes, mejoras y planeación del siguiente periodo.
- **Planeamiento Blitz:** En este juego, se ponen tarjetas indexadas en una mesa, con una historia de usuario o función visible en cada una. El grupo finge que no hay dependencias entre tarjetas, y las alinea en secuencias de desarrollo preferidas. Los programadores escriben en cada tarjeta el tiempo estimado para desarrollar cada función. El patrocinador del usuario escribe la secuencia de prioridades, teniendo en cuenta los tiempos referidos y el valor de negocio de cada función. Las tarjetas se agrupan en períodos de tres semanas llamados iteraciones que se agrupan en entregas, usualmente no más largas de tres meses.
- **Estimaciones Delphi:** Se reúnen los expertos responsables y proceden como en un remate para proponer el tamaño del sistema, su tiempo de ejecución, la fecha de las entregas según dependencias técnicas y de negocios y para equilibrar las entregas en paquetes de igual tamaño.
- **Encuentros diarios de pie**. La palabra clave es “brevedad”, cinco a diez minutos como máximo. No se trata de discutir problemas, sino de identificarlos.
- **Gráficos de quemado:** Se trata de una técnica de graficado para descubrir demoras y problemas tempranamente en el proceso, evitando que se descubran demasiado tarde. Para ello se hace una estimación del tiempo faltante para

programar lo que resta al ritmo actual, lo cual sirve para tener dominio de proyectos en los cuales las prioridades cambian bruscamente y con frecuencia. Los gráficos de quemado ilustran la velocidad del proceso, analizando la diferencia entre las líneas proyectadas y efectivas de cada entrega.

- **Programación lado a lado:** la versión de Crystal Clear establece proximidad, pero cada quien se enfoca a su trabajo asignado, prestando un ojo a lo que hace su compañero, quien tiene su propia máquina.

### 4.4.4. Metodología DSDM(Dynamic Systems Development Method).

Principios:

1. El involucramiento del usuario es obligatorio.
2. Los equipos de DSDM deben tener el poder de tomar decisiones.
3. la atención está puesta en la entrega frecuente de productos.
4. La conformidad con los propósitos del negocio es el criterio esencial para la aceptación de los entregables.
5. El desarrollo iterativo e incremental es necesario para converger hacia una correcta solución del negocio.
6. Todos los cambios durante el desarrollo son reversibles.
7. Los requerimientos están especificados a un alto nivel.
8. Las pruebas están integradas a través del ciclo de vida.
9. Un enfoque colaborativo y cooperativo entre todos los interesados es esencial.

Fases de construcción del sistema:

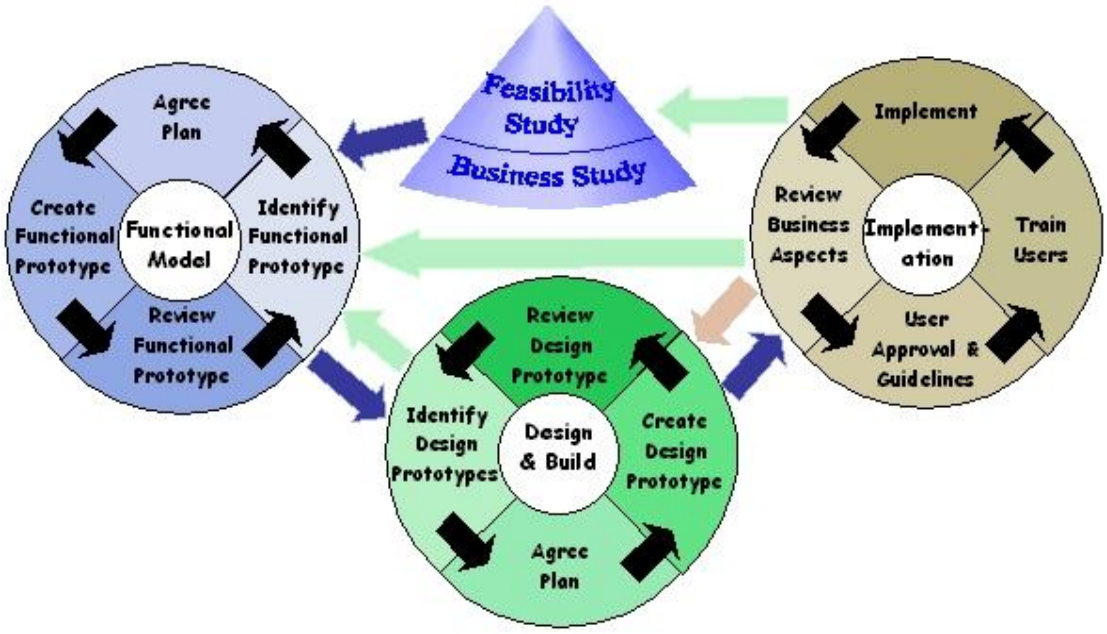
1. Estudio de factibilidad
2. Estudio del negocio
3. Iteración del modelo funcional
4. Iteración del diseño
5. Construcción

6. Implantación

En DSDM, un timebox consta de tres fases que son:

- **Investigación:** Se chequean que las actividades que componen el timebox se coincidan con la arquitectura del sistema. Esta es una fase de carácter exploratorio, en la que se fijan los objetivos de la iteración, los entregables a ser producidos, efectuándose revisiones sobre las iteraciones anteriores a la actual.
- **Refinamiento:** Consiste en la producción propiamente dicha de los artefactos planificados
- **Consolidación:** Consiste en completar los entregables, verificando la calidad de los mismos

**Figura 3. Proceso DSDM**[5]




[5] Imagen tomada de: What is DSDM?. [Artículo en internet]. http://www.codeproject.com/KB/architecture/dsdm.aspx [Consultado en Octubre del 2011]

Roles:

- **Visionario:** Encargado de asegurar que se satisfacen las necesidades del negocio
- **Usuario Embajador:** Brinda el conocimiento del negocio y define los requerimientos del software.
- **Coordinador técnico:** Persona encargada de mantener la arquitectura y verificar la consistencia de los componentes construidos respecto a esta y al cumplimiento de los estándares técnicos.

## 4.5. METODOLOGÍAS MIXTAS O SIMPLIFICADAS

### 4.5.1. AUP (Agile Unified Process)

Es una versión simplificada del proceso RUP(Rational Unified Process), describe de manera fácil y simplificada la forma de abordar proyectos de software aprovechando técnicas ágiles de las cuales carece RUP para mejorar la productividad.

Características:

- **El personal sabe lo que está haciendo:** No se tiene documentación estricta, pero si hay guías y se hacen entrenamientos que permiten estar al tanto del proyecto.
- **Sencillez:** Todo se describe de manera concisa en unas cuantas páginas y no en miles de ellas
- **Agilidad:** Se siguen los valores y principios del manifiesto ágil y la alianza ágil.
- **Enfoque en las actividades de alto valor**: Se enfoca en las actividades que cuentan y no en las posibles cosas que puedan suceder en el proyecto
- **Independencia de las herramientas de trabajo:** No importa que herramientas se usen para hacer el trabajo
- Se puede adaptar el producto para satisfacer necesidades individuales : no necesita ninguna herramienta especializada para adaptar AUP.

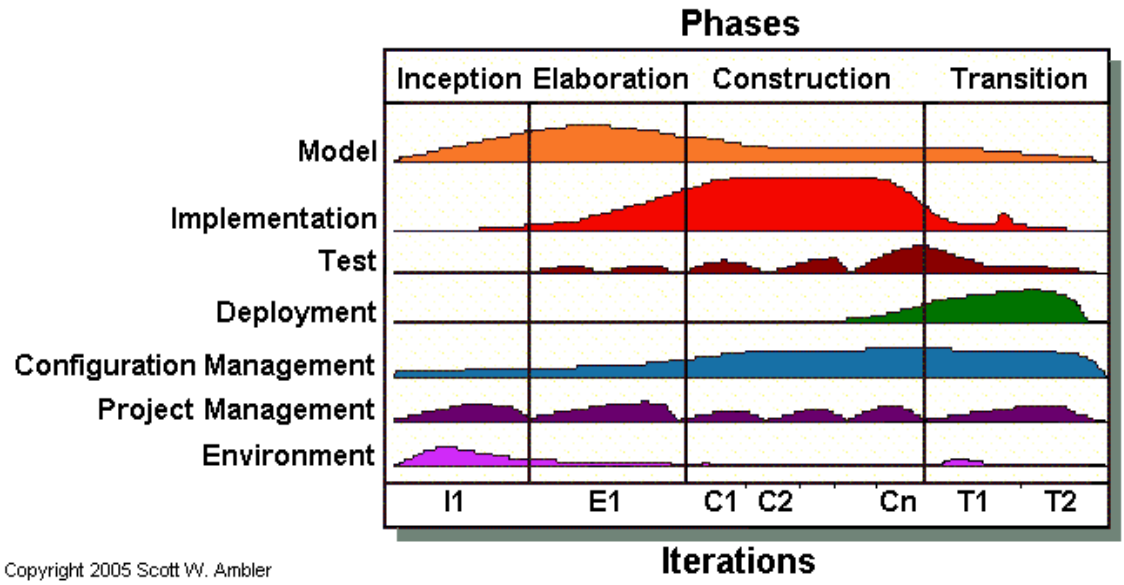
### Roles:

- Diseñador BD
- Modelador
- Manejador de la configuración
- Desarrollador
- Encargado del despliegue
- Ingeniero de procesos
- Administrador del proyecto
- Encargado de revisar el progreso
- Interesados
- Encargado de la documentación
- Administrador de pruebas
- Encargado de las pruebas
- Especialista en herramientas

### Fases o ciclo de vida:

1. **Inicio:** El propósito es identificar el alcance inicial del proyecto, una posible arquitectura para el sistema, obtener los fondos para el proyecto y la aceptación de los interesados
2. **Elaboración:** Se plantea y aprueba la arquitectura del sistema
3. **Construcción:** Se construye el sistema
4. **Transición:** El propósito es validar y desplegar el sistema.

**Figura 4. Ciclo de vida AUP**[6]




Disciplinas:

1. **Modelo:** En esta disciplina se trata de entender el dominio del problema, la lógica de negocio y la posible solución al problema
2. **Implementación:** El propósito es transformar el modelo en código ejecutable, con un testeo básico de funcionamiento.
3. **Pruebas:** Se hacen para evaluar la calidad, encontrar defectos, validar que el sistema funcione como se planteó y verificar que los requisitos sean conocidos.
4. **Despliegue:** El propósito es planear la entrega del sistema y ejecutar el plan para que el sistema esté disponible para los usuarios
5. **Manejo de la configuración:** Para manejar el acceso a los artefactos del proyecto, versionamiento y manejo de cambios.
6. **Administración del proyecto:** coordinar las actividades, manejar riesgos, dirigir y coordinar personal, recursos y entregas.

---

[6] Imagen tomada de: The Agile Unified Process (AUP). [Artículo en Internet]. *http://www.ambysoft.com/unifiedprocess/agileUP.html* [Consultado en Septiembre del 2011]

7. **Ambiente de trabajo:** Asegurar que las herramientas, el ambiente de trabajo y las guías estén disponibles cuando se necesiten.

### 4.5.2. Scrum wrapper for XP

Se trata de unir estas SCRUM y XP para crear una metodología ágil más robusta y que combine las virtudes de ambos.

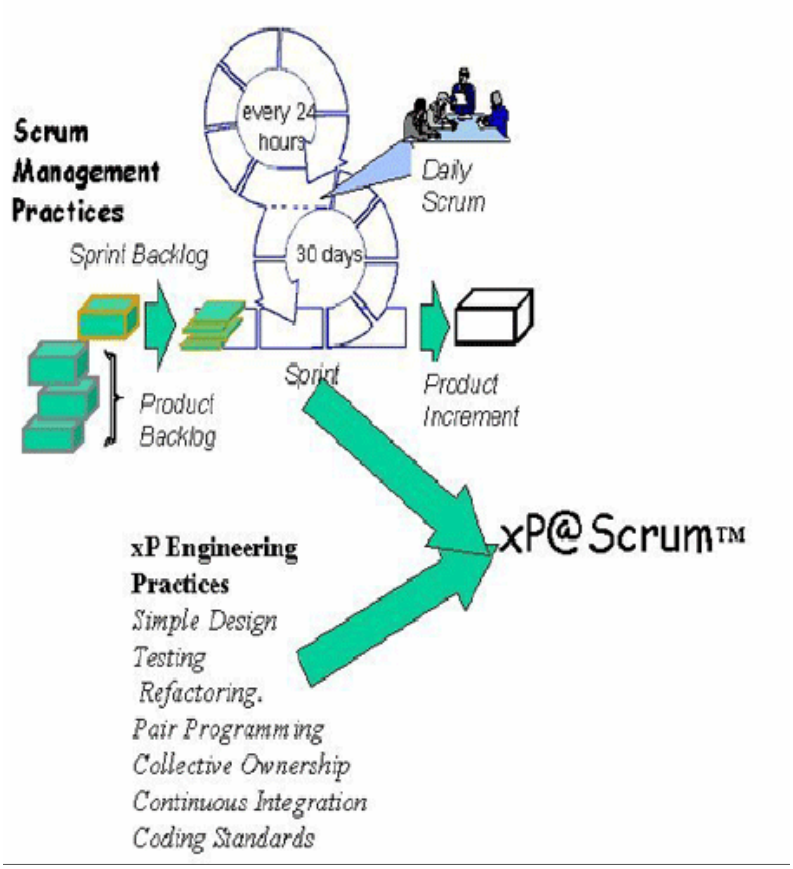
#### Características:

- SCRUM como un contenedor de gestión para las prácticas de ingeniería de XP
- SCRUM provee los mecanismos de administración ágil
- XP aporta las prácticas integradas de ingeniería

#### Beneficios:

- Los mecanismos de administración y control de SCRUM pueden ser aplicados para cualquier tipo de proyecto
- Los proyectos que cuentan con esta combinación se vuelven escalables y pueden ser realizados en simultaneo con equipos que no comparten una misma ubicación geográfica
- SCRUM se implementa en un día, XP se puede implementar gradualmente dentro de SCRUM
- Esta combinación ha demostrado escalabilidad lineal en proyectos distribuidos, de contratación externa y CMMI nivel 5.

**Figura 5. Scrum wrapper for XP**[7]




## 4.5.3. OPENUP

Es una metodología para el desarrollo de software basado en RUP, que aplica las ventajas del desarrollo incremental e iterativo en su ciclo de vida, OPENUP tiene un enfoque de filosofía ágil, teniendo en cuenta la naturaleza colaborativa del desarrollo de software, no está basada en ninguna herramienta específica y no incluye un proceso extenso.

[7] Imagen tomada de: Scrum with XP and Beyond. [Documento en Internet]. http://www.gbcacm.org/sites/www.gbcacm.org/files/slides/7A%20-%20Jeff%20on%20Scrum%20and%20XP.pdf [Consultado en Septiembre del 2011]

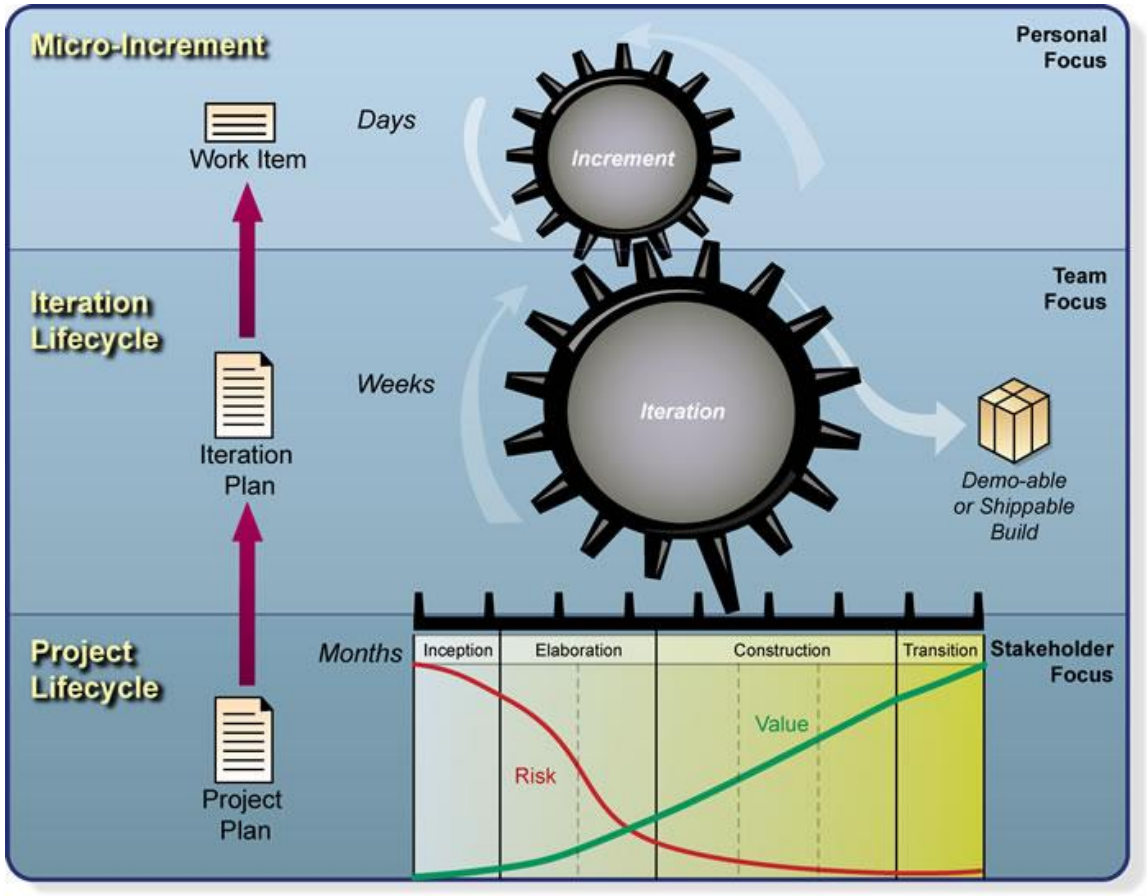
### Principios:

- Equilibrar las prioridades para maximizar el valor que obtienen los interesados: promover prácticas que permitan a los participantes del proyecto y los interesados desarrollar una solución que maximice los beneficios de los interesados que cumpla con los requisitos y restricciones establecidas en el proyecto.
- Colaborar para sincronizar intereses y compartir conocimiento: promover prácticas que fomenten un buen ambiente de trabajo, permitan la colaboración y desarrollen un entendimiento compartido del proyecto
- Enfocarse en la arquitectura de forma temprana para minimizar el riesgo y organizar el desarrollo
- Desarrollo evolutivo, para obtener realimentación y mejora continua.

### Fases o ciclo de vida:

- Inicio
- Elaboración
- Construcción
- Transición

**Figura 6. Ciclo de vida OpenUP**[8]




## Roles:

- Analista
- Arquitecto
- Desarrollador
- Administrador del proyecto
- Interesados
- Tester

[8] Imagen tomada de: OpenUP [Artículo de Internet]. 2011. http://epf.eclipse.org/wikis/openup/ [Consultado en Septiembre del 2011]

## 4.6. CASOS DE USO EXITOSOS USANDO METODOLOGÍAS ÁGILES

### 4.6.1. Primavera Systems, Inc.

A sus 21 años de edad, esta empresa vende soluciones para el manejo del portafolio de productos, ayudando a las empresas a gestionar todos sus proyectos, programas y recursos. Primavera estaba prosperando y creciendo, sus clientes estaban aumentando la complejidad de sus necesidades; esto puso a prueba la habilidad de liberar productos que cubrieran las necesidades de sus clientes. Durante el 2002 los desarrolladores trabajaron sobre el tiempo para liberar la versión 3.5 de su software, al igual que con otros productos en el pasado, los últimos 3 meses eran particularmente difíciles, pues el personal sacrificaba fines de semana y vida familiar para conseguir los objetivos. El resultado fue una versión incompleta, con 3 semanas de retraso y un equipo de trabajo exhausto y con la moral baja.

Primavera decidió probar las metodologías ágiles, particularmente SCRUM y XP. Al comienzo utilizaron SCRUM para mejorar su proceso de desarrollo, después empezaron a adoptar prácticas de XP para mejorar la calidad de sus productos y la eficiencia, finalmente fueron personalizando su proceso para adaptarse a sus propias necesidades. Los resultados fueron unos clientes satisfechos y un entorno de trabajo altamente motivado y enérgico.[9]

### 4.6.2. Workshare Technology.

Workshare es una empresa que se mueve en el campo de trabajo de la innovación y la gestión de documentos técnicos. Debido a que tienen productos únicos y se encuentra en un mercado de ritmo rápido y constante cambio, sus diseños y

[9] Caso de éxito tomado de: Best Practices in Scrum Project Management and XP Agile Software Development [Artículo de Internet] http://www.objectmentor.com/resources/articles/Primavera.pdf [Consultado en Octubre de 2011]

especificaciones funcionales necesitan ser extremadamente flexibles, casi tan rápidos como se escriben, las especificaciones se convirtieron en algo obsoleto y los cambios recientes se encuentran en la cabeza del personal, si bien sus métodos eran ágiles, no eran escalables para el crecimiento y las necesidades del equipo de ingeniería, se presentaban los siguientes problemas:

- El desarrollo era conducido por los "Bugs" en vez de las necesidades funcionales
- Los conocimientos de programación no eran compartidos por igual, se tenían demasiados especialistas con poco tiempo para entrenar al nuevo personal
- A pesar de haber pruebas unitarias, eran excepciones en vez de ser la regla
- Se estaba construyendo software frecuentemente, pero poniendo la garantía del control de la calidad en detectar los errores y defectos dentro del código, en vez de construir con calidad.

Workshare necesitaba una metodología que pudiera ser usada para el despliegue rápido de software, mejorar la comunicación entre todos los departamentos, ser escalable y encajar en la cultura y filosofía de la organización. XP cumple con todos estos requisitos y se convirtió en la metodología utilizada por esta empresa.[10]

### 4.6.3. The Combat Identification Server (CIdS) Technology Demonstrator Project (TDP)

CIdS TDP ha sido fundado por MoD DE&S Tactical Datalinks Delivery Team (TDL DT) y entregado por un consorcio industrial liderado por General Dynamics United Kingdom Limited. CIdS es un complejo sistema – proyecto de software con el objetivo de ayudar a eliminar "la niebla de la guerra" ofreciendo una imagen en la cabina de una aeronave con la posición de sus fuerzas aliadas en tierra.

---

[10]Caso de éxito tomado de: Workshare Technology and eXtreme Programming (XP) [Artículo de Internet] http://www.objectmentor.com/resources/articles/workshare.zip [Consultado en Octubre de 2011]

El uso de DSDM fue una decisión audaz, pero el proyecto ha sido entregado rápidamente con resultados satisfactorios para todos los involucrados teniendo en cuenta que en el actual contexto económico, la entrega de soluciones dentro del presupuesto es la máxima prioridad.

Mediante el uso de DSDM en el proyecto, la tradicional división entre cliente / proveedor y contratista / subcontratista han sido integrados para formar un verdadero equipo integrado de trabajo, el todos para uno y uno para todos prevaleció a lo largo del desarrollo, resultando un equipo enfocado en la entrega de los objetivos del proyecto en lugar de lo que es mejor para cada organización individual.[11]

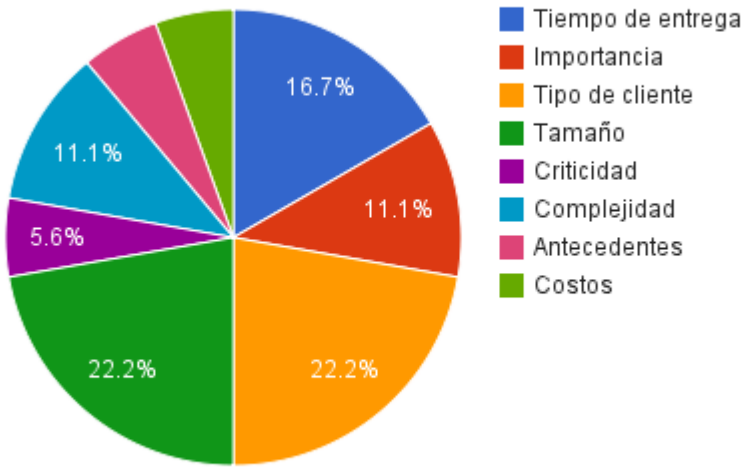
[11]Caso de éxito tomado de: Improving Outcomes through Agile Project Management [Artículo de Internet] *http://www.dsdm.org/wp-content/uploads/2011/02/Improving-Outcomes-Through-Agile-Project-Management.pdf* [Consultado en Octubre de 2011]

# 5. ANÁLISIS DE LAS METODOLOGÍAS DE DESARROLLO DE SOFTWARE UTILIZADAS A NIVEL REGIONAL

Para tener una idea de cuales metodologías y prácticas están utilizando las empresas del departamento de Antioquia para el desarrollo de software se realizó una encuesta la cual arrojó los siguientes resultados.

¿Al momento de escoger una metodología para desarrollar un proyecto qué factores se tienen en cuenta?

**Figura 7. Factores tenidos en cuenta para la escogencia de una metodología de desarrollo**




De acuerdo a la encuesta, el enfoque local para la elección de metodologías para el desarrollo de software está basado principalmente en las características inherentes al

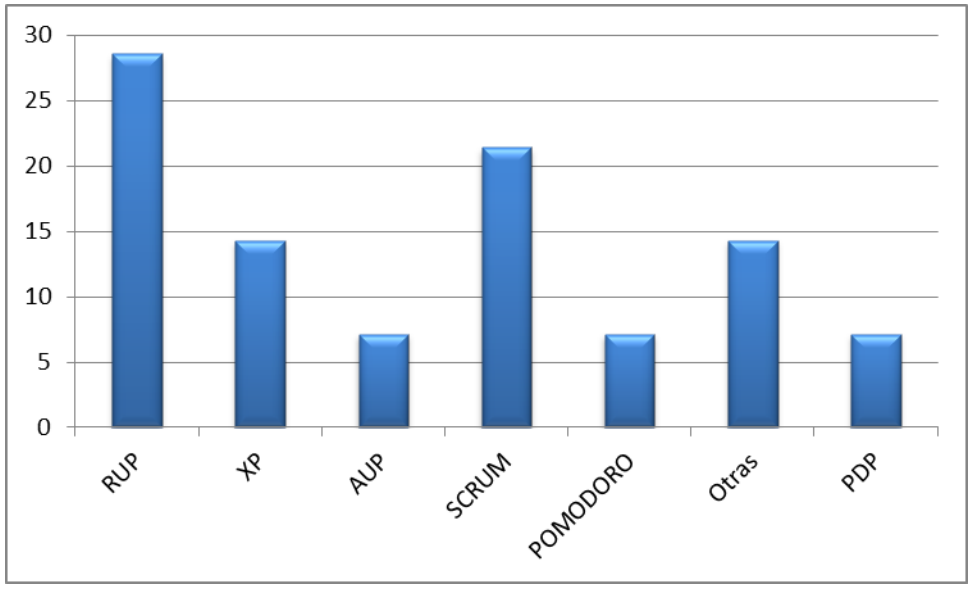
producto que se planea crear en el proyecto, tales como: tamaño y lógica de negocio plasmada en él.

También podemos destacar que otro de los factores que más se tienen en cuenta tiene que ver con el tiempo de entrega del proyecto, la importancia con la que se programa y la complejidad involucrada para desarrollarlo.

En conclusión y teniendo presente que la respuesta de las empresas al preguntar si se usaban varias metodologías fue NO el 70% de los casos, en el medio están interesado en escoger una metodología pensando en la robustez y la capacidad de manejar toda la información que los proyectos implican, además de no olvidar el tiempo en que son llevados a cabo los desarrollos. Parece claro que el costo, los antecedentes y la criticidad del proyecto no han sido factores clave a la hora de escoger la forma de abordar los proyectos.

¿Qué metodologías o en cuales se basan las empresas para el desarrollo de los proyectos de software?

**Figura 8. Metodologías base utilizadas por las empresas Antioqueñas**




De acuerdo al gráfico anterior la gran mayoría de las empresas de Antioquia se basan en RUP para definir sus metodologías de desarrollo de software, sin embargo es de destacar que en el medio hay una alta percepción de la importancia de mejorar sus metodologías para dotarles de la flexibilidad y velocidad que la vertiente ágil puede significarles.

Se hace notoria la importancia de RUP como metodología propuesta por IBM, pues es una metodología altamente reconocida y promovida en el medio, claro está, que al ser avalada por una empresa con tal renombre y experiencia en el medio, genera la confianza y estabilidad que las organizaciones necesitan para el desarrollo de sus productos de software.

## 5.1. JUSTIFICACIÓN DE LA ENCUESTA

La información contenida en la encuesta que se hizo a varias personas pertenecientes al sector de desarrollo de software de la región de Antioquia tiene como propósito conocer a grandes rasgos que metodologías se están utilizando a la hora de elaborar software en el entorno empresarial. Estos datos nos presentan la forma en que son abordados los desarrollos y las problemáticas en la región, permitiendo proponer una metodología basada en las prácticas y métodos comúnmente usados en nuestra industria, además de añadir la filosofía ágil sin negar el conocimiento obtenido en el sector.

## 5.2. RUP

Rational unified Process, es una metodología propuesta por IBM, y basada en las mejores prácticas del mercado. Es una metodología incremental y con un ciclo de vida en espiral, se divide en fases y se basa en iteraciones de estas.

### Prácticas:

- **Desarrollo Iterativo de software:** Entre las iteraciones se van agregando nuevas funcionalidades y mejorando las ya existentes.

- **Manejo de requisitos:** RUP describe todo el manejo de requisitos, desde la elicitación y organización hasta la documentación.

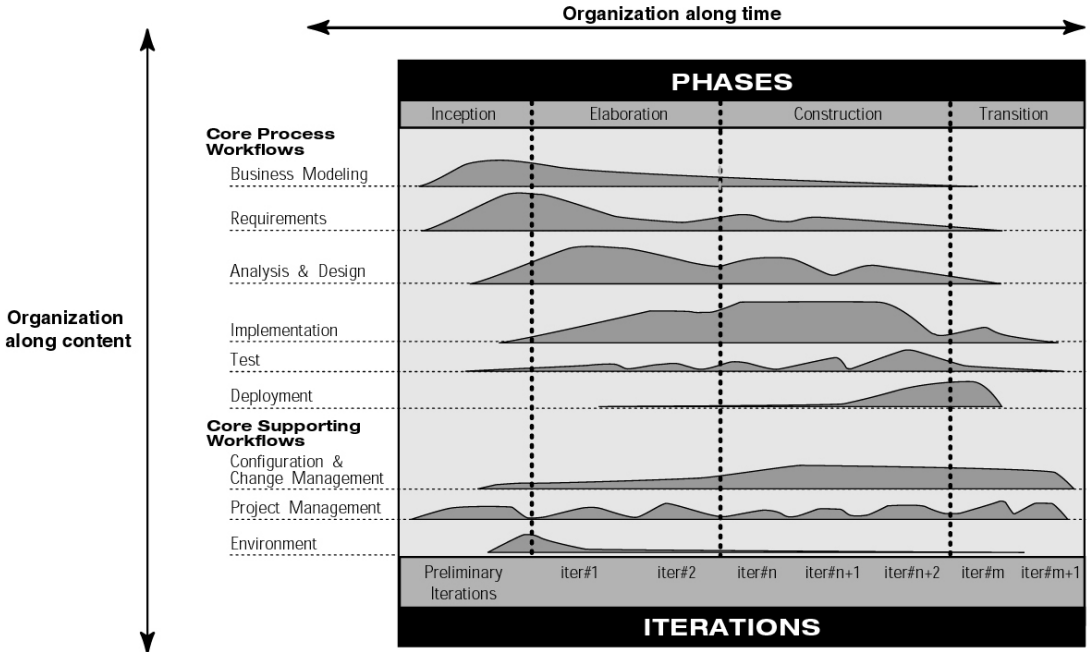
- **Uso de arquitecturas basadas en componentes:** La arquitectura debe ser flexible, entendible y promover la reutilización, esta metodología propone que sea orientada a componentes para que se cumpla con este cometido.

- **Modelo Visual del software:** Se promueve el uso de UML como lenguaje para describir el funcionamiento del software

- **Verificación de la calidad del software:** RUP presta asistencia para la planeación y ejecución de pruebas de aseguramiento de calidad.

- **Control de cambios sobre el software:** El desarrollo de software siempre está envuelto e cambios de funcionalidad y requisitos, RUP describe el proceso a seguir para rastrear y ejecutar un control sobre las nuevas especificaciones y funcionalidades que se presentan.

**Figura 9. El proceso RUP[12]**




## Ventajas metodología RUP

- Evaluación de cada fase que permite cambios de objetivos
- Funciona bien en proyectos de innovación
- Seguimiento detallado de cada una de las fases

[12] Imagen tomada de: Rational Software White Paper TP026B, Rev 11/01. [Artículo en Interner] *http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf* [Consultado en Septiembre del 2011]

- Permite evaluar tempranamente los riesgos
- Al realizarse revisiones en las primeras iteraciones permite reutilizar código y mejorar el ya existente
- Altamente utilizada en el medio
- Bueno en proyectos grandes

### Desventajas metodología RUP

Al no ser una metodología ágil RUP tiene las siguientes desventajas:

- Centrada en Riesgos, además tiene un complejo proceso para su seguimiento y control.
- El proceso de desarrollo es complejo y desorganizado
- Alta generación de documentación
- Poco centrada en las personas
- Proceso demasiado largo para la obtención del producto final.
- En proyectos pequeños puede no ser rentable

## 5.3. RUP COMO BASE PARA LA CREACIÓN DE UNA METODOLOGÍA MÁS ÁGIL

Para definir una metodología de software tomaremos como base lo planteado por Alistair Cockburn[13] quien propone que una metodología se puede desglosar en 10 elementos: Roles, Destrezas, Actividades, Entregables, Técnicas, Valores, Equipos, Asignación de tareas, Herramientas, Estándares.

[13] Cockburn, Alistair, Surviving Object Oriented Projects, Addison-Wesley Object Technology Series, 1998.

Para efectos de definición de la metodología propuestas solo tendremos en cuenta los elementos principales: Actividades, Roles, Entregables, Técnicas, Valores.

Después de exponer las ventajas y desventajas de RUP como proceso de desarrollo de software, se tomarán algunas de sus características y se agregarán otras tomadas de las metodologías ágiles presentadas al comienzo de este trabajo para el planteamiento de una metodología alterna que logre mejorar la velocidad con que se crea software en las empresas regionales.

# 6. DEFINICIÓN DE UNA METODOLOGÍA TRANSVERSAL BASADA EN LAS NECESIDADES LOCALES Y LA APLICACIÓN DE TÉCNICAS ÁGILES.

## 6.1. CARACTERÍSTICAS

La metodología propuesta está orientada al desarrollo ágil, iterativo e incremental, busca garantizar que se generen solo los artefactos mínimos evitando hacer mal uso del tiempo como recurso más importante e irreemplazable. También tiene en cuenta la necesidad de hacer entregas funcionales en corto tiempo, pues para los interesados es importante ver cómo está evolucionando el proyecto de forma tangible y no solo a base de informes y diseños.

La comunicación juega un papel importante en esta metodología pues al haber poca documentación escrita del proyecto, se hace necesario un ambiente de trabajo abierto no solamente en el que se comuniquen dificultades y avances sino también en el que se compartan ideas y soluciones.

Esta metodología está dirigida a equipos de trabajo pequeños de máximo 7 personas y busca potenciar las habilidades de desarrollo de software para obtener resultados rápidos y deseados, pues soporta el cambio de requisitos en cualquier etapa y no está sujeta rigurosamente a lo planeado, además busca generar resultados en corto tiempo.

Se debe poner especial hincapié en la disciplina, pues toda metodología ágil conlleva a la ejecución de sus prácticas y al cumplimiento en la duración que se establezca para las iteraciones del proyecto.

## 6.2. FASES Y ACTIVIDADES

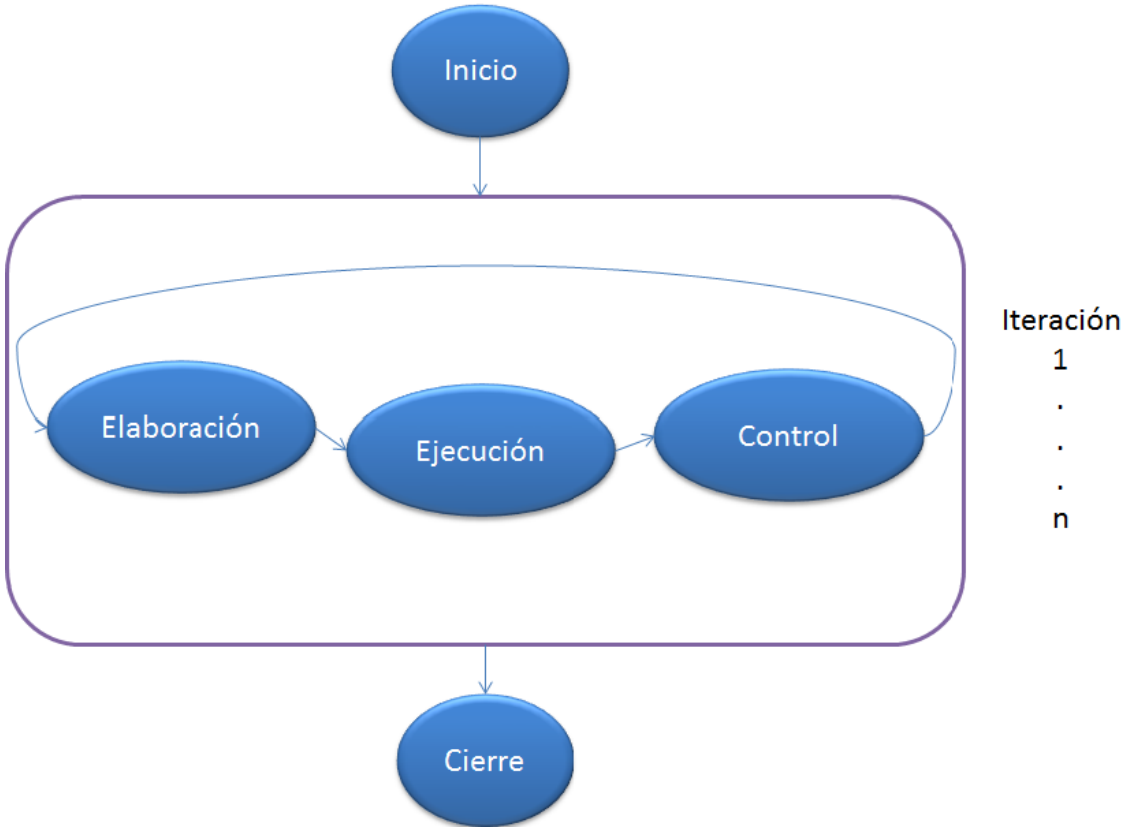
Se plantean 3 Fases:

- **Inicio:** En esta fase se define una visión inicial del proyecto, su factibilidad, el presupuesto, el número de iteraciones, los requisitos, los casos de uso de alto nivel y se propone una arquitectura base.
- **Iteración:** De acuerdo a las iteraciones propuestas o a las necesidades que surjan se ejecutan las siguientes sub-fases por cada iteración.
    - **Elaboración:** Se ejecuta el análisis y diseño para la iteración, se establece el producto funcional de resultado, se propone una solución para el conjunto de requisitos incluidos, se establecen tareas y tiempos.
    - **Construcción:** Se centra en la elaboración del subproducto u objetivo propuesto para la iteración
    - **Control:** Se entrega el resultado del trabajo para que el cliente pueda ver y evaluar su funcionamiento, se puede implantar la solución obtenida, se entrena a los usuarios.

- **Cierre:** Se da por terminado el proyecto

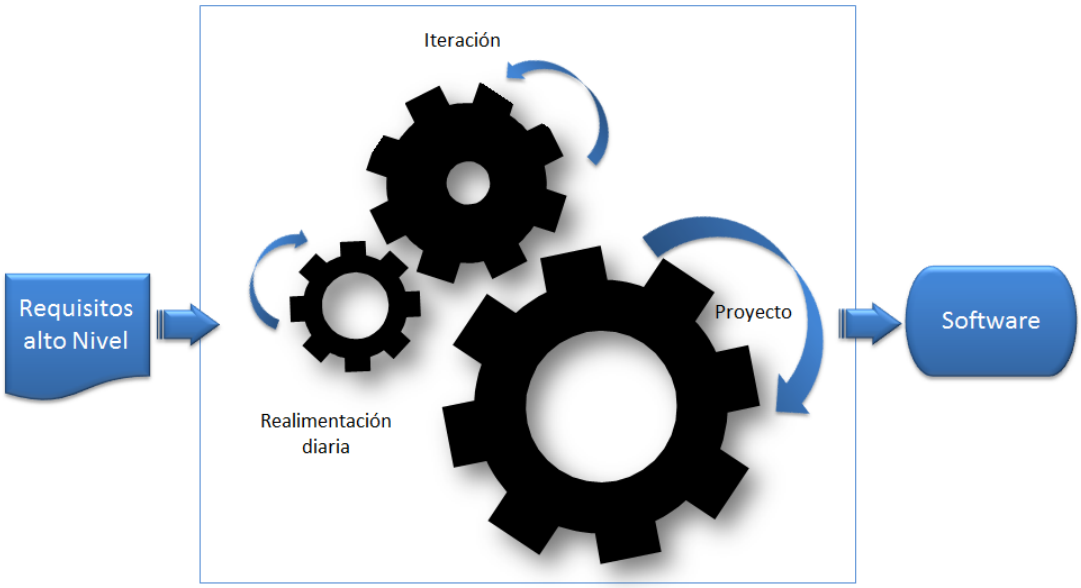
El cumplimiento de los requisitos establecidos para cada iteración se hace crucial para el éxito del proyecto, pues esta metodología implica una gran disciplina para lograr los objetivos propuestos al inicio de cada iteración.

**Figura 10. Proceso metodología propuesta**




Al igual que en RUP, las iteraciones se hacen comenzando desde la elaboración hasta la entrega de un resultado, volviendo a ejecutarse para la próxima iteración este mismo ciclo.

**Figura 11. Ciclo de vida metodología propuesta**




## 6.3. ROLES

- **Usuarios del sistema:** Interesados o afectados por el producto, tienen como responsabilidad proveer el conocimiento del negocio, los requisitos y la realimentación de los resultados del producto obtenido.

- **Dueño del producto:** Es el encargado de proveer los recursos financieros. Junto al administrador del proyecto es quien establece los tiempos de entrega, la cantidad de iteraciones, el contenido de las iteraciones y aprueba los productos obtenidos.

- **Administrador del proyecto:** Es el responsable del proyecto, debe coordinar al equipo de trabajo y estar en constante comunicación con el dueño del producto, es el intermediario y encargado de la buena comunicación entre todos los implicados en el proyecto.

- **Equipo de trabajo:** Debe ser poli-funcional y auto-organizado, es quien lleva a cabo la realización del producto.

Al contar con una cantidad mínima de roles, la metodología propuesta permite que tanto las actividades como el conocimiento sean compartidos, logrando flexibilidad para la asignación de tareas y una visión general del proyecto para todos los integrantes del equipo. Además es importante destacar que al disminuir la jerarquía se puede lograr un verdadero trabajo en equipo, ahorrándose el costo tanto en tiempo como en recursos de escalar a través de distintos niveles de autoridad. La única figura de jerarquía corresponde al dueño del producto.

**Figura 12. Roles y comunicación**


## 6.4. ENTREGABLES

**Fase de Inicio:**

- Cronograma
- Descripción general del sistema

- Arquitectura base propuesta (generalidades base de datos y clases)
- Presupuesto
- Lista de requisitos de alto nivel

Por cada iteración en cada sub-fase se harán las siguientes entregas:

**Elaboración:**

- Cronograma detallado de la iteración
- Casos de uso
- Diagrama de Base de Datos
- Diagrama de clases
- Pantallazos del sistema (Prototipos no funcionales)

**Ejecución:**

- Código fuente ejecutable

**Control:**

- Documento de realimentación (cambios y mejoras)
- Aplicación funcional

## 6.5. VALORES

- **El trabajo en equipo y la comunicación como pilar para el entendimiento y solución de problemas:** En lo posible cada persona debe estar al tanto de lo que sucede en su entorno, así como de las dificultades y soluciones que los demás han hallado permitiendo la reutilización de soluciones y evitando la reinvención de la rueda.

- **El código debe documentarse por sí mismo:** Para evitar que se generen grandes cantidades de documentación que al final no se usará. Se hace necesario que el código sea entendible y auto explicable. El nombramiento de las variables, métodos y clases usadas deben estar acorde a la lógica de la aplicación.

- **Los riesgos suelen materializarse con el tiempo, si hay conciencia de lo que pasa habrá solución en el momento de presentarse:** Cuando se vive todo el proceso de desarrollo de software, se crea una visión general que permite tener una solución a todo riesgo que pueda materializarse.

- **Los requisitos siempre cambiarán:** No hay software que no esté sujeto a cambios, siempre debe ser posible modificar su funcionamiento para que satisfaga la necesidad actual.

- **El aprendizaje está limitado por la habilidad de una organización a mantener su gente:** Se debe garantizar un buen ambiente de trabajo y estabilidad al equipo de trabajo, pues los cambios de personal involucran pérdida de conocimiento obtenido por el proyecto.

## 6.6. TÉCNICAS PROPUESTAS

- **Máxima comunicación:** Se recomienda que la comunicación en el equipo sea cara a cara pues esto aumenta la capacidad de entendimiento, la resolución de problemas y el surgimiento de conocimiento. En última instancia sería posible trabajar con la metodología propuesta de forma remota si hay comunicación vía voz. No se recomienda que el grupo esté disperso en más de 2 ubicaciones geográficas.

- **Reuniones cortas tipo SCRUM:** Se llevan a cabo diariamente, aunque puede acordarse con el equipo para hacerlas con una periodicidad diferente. Es

importante que se ejecute en el mismo lugar, a la misma hora y que su duración no sea mayor a 30 minutos. A cada miembro del equipo se le hacen 3 preguntas respecto a su última situación antes de la reunión: qué hizo, qué obstaculizó su trabajo y qué planea hacer. Sobre las respuestas y sus problemas asociados pueden programarse reuniones de subgrupo para darles solución.

- **Interrupciones mínimas:** Se debe evitar que el equipo esté localizado en un área de trabajo en donde pueda ser interrumpido de sus labores establecidas con información de otras áreas de la organización.

- **Radiadores de Información:** Si es posible contar con espacios para colgar tableros, diagramas o información importante referente al proyecto, esto permite tener información de manera fácil. También es posible usar otros medios como páginas web o repositorios.

- **Disponibilidad de usuarios del sistema para la resolución de dudas:** en lo posible contar con un usuario experto del sistema que pueda dar respuesta a las dudas en cuestión de horas.

- **Pruebas:** Producción de código dirigida por pruebas unitarias y de integración que son ejecutadas constantemente ante cada modificación del sistema.

- **Refactorización:** Actividad constante de reestructuración del código con el fin de remover duplicación de código, incrementar legibilidad, simplificarlo y hacerlo más flexible a futuros cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo.

- **Estándares de programación:** La comunicación de los programadores es a través del código. Se debe seguir ciertos estándares para mantener legible el código.

- **Entrevistas de proyectos:** se debe entrevistar a más de un responsable para tener visiones más ricas de la situación.

- **Programación lado a lado:** debe haber proximidad entre el equipo, pero cada quien se enfoca a su trabajo asignado, “prestando un ojo” a lo que hace su compañero, quien tiene su propia máquina.

- **Propiedad colectiva del código:** Cualquier programador puede cambiar cualquier parte del código en cualquier momento.

## 6.7. ASPECTOS A TENER EN CUENTA

La implantación de cualquier metodología ágil implica un gran esfuerzo debido a que todo cambio es difícil. Dependiendo de la metodología utilizada se puede comenzar a usar las técnicas de forma gradual, pero para la parte del proceso se recomienda que sea ejecutada desde el inicio del proyecto para tener buenos resultados.

La metodología propuesta no está pensada para equipos y proyectos grandes pues a medida que crece el número de personas involucradas en el desarrollo del mismo, la comunicación se hace más difícil y sería necesario utilizar otros canales de comunicación y técnicas para lograr el orden y el desarrollo de los proyectos.

# 7. HERRAMIENTAS PROPUESTAS

Para utilizar la metodología mencionada no es necesario la dependencia de alguna herramienta, se pueden utilizar aquellas con las que hay más familiaridad, bien sean de pago o open source. Sin embargo, se recomienda hacerse de unas buenas herramientas pues gran parte del trabajo depende del uso preciso de los recursos técnicos.

A continuación se presenta una breve recopilación de herramientas que pueden ser útiles para la implementación de la metodología propuesta. Su uso no es obligatorio ni son las únicas en el mercado, pero si es destacable su utilidad.

**CVSNT:** Es un software para el manejo de versiones. Acompañado de sus respectivos programas cliente, sirve tanto para guardar archivos como para llevar un control sobre los cambios en el código del sistema que se está desarrollando. Puede ser utilizado para el trabajo en equipo, centralizando el código en un servidor y permitiendo cambios desde diferentes puestos de trabajo.

Es una herramienta indispensable en la etapa de construcción del sistema, pero también puede ser utilizada como soporte para los documentos y archivos generados en las otras etapas.

**TortoiseCVS:** Es un cliente CVS que trabaja sobre el sistema operativo Windows. Es el complemento perfecto de CVSNT para el manejo de documentos, gráficos y demás archivos que se generan en el desarrollo de software.

**JUnit:** Es un framework que permite ejecutar pruebas automáticas sobre las clases java que se estén desarrollando. Puede ser de gran utilidad durante la etapa de construcción para probar el código que se va creando.

**CheckStyle:** Es una herramienta de desarrollo que ayuda a los programadores a escribir código Java de acuerdo a unos estándares establecidos. Se puede automatizar el proceso de chequear el código escrito para evitar hacer manualmente este aburrido proceso y obtener grandes ventajas en el entendimiento del mismo.

**AndroMDA:** Es un framework generador de código extensible que permite a partir de un modelo UML generar artefactos en Java. Se debe tener en cuenta que funciona mejor sobre proyectos en donde:

- Se comienza desde cero
- Se quiere ahorrar tiempo generando código
- Se quiere el almacenamiento sobre base de datos
- Se usa UML

No es recomendable para los proyectos en donde:

- Se usa una base de datos existente que no puede ser fácilmente mapeada a objetos java
- Se trabaja sobre una aplicación madura y no es fácil modelar los componentes existentes.
- No se usan herramientas para la integración continua, pruebas unitarias, reutilización de código.

# 8. CONCLUSIONES

- Las metodologías tradicionales seguirán siendo importantes durante mucho tiempo, pues son muy útiles para proyectos de gran tamaño, alto riesgo y que implican equipos de trabajo numerosos y dispersos geográficamente. Sin embargo, las metodologías ágiles siguen ganando fuerza pues evitan tener que ejecutar muchas actividades innecesarias al desarrollar proyectos pequeños y medianos que pueden ser llevados a cabo por equipos con menor número de miembros localizados en un mismo sitio y guiados por las técnicas que las metodologías ágiles proponen.

- Cualquiera sea la metodología utilizada por alguna organización o grupo de desarrolladores de software, esta se debe ir adaptando y alimentando con las necesidades específicas y conocimiento adquirido por quienes hacen uso de ella. Es importante que las metodologías evolucionen tal como lo hacen los sistemas, reforzándose con nuevas herramientas y con prácticas más acordes al medio donde se utilizan.

- La experiencia y el pasar de los años sobre las ciencias de la computación han demostrado que los ciclos de vida más acordes para agilizar un proyecto de software de pequeño y mediano tamaño con requisitos cambiantes son incrementales e iterativos, pues permiten a medida que se va avanzando, ir revisando y mejorando el sistema, por tanto es posible realizar modificaciones aun en etapas avanzadas sin altos costos ni mayores complicaciones.

- Los mecanismos para el control de cambio, manejo de riesgos y documentación del proyecto son muy importantes en aquellos proyectos en donde los equipos no trabajan de forma unida pues la información no es tan fácil de transmitir y cada persona puede estar centrada en sus actividades. Es por eso que en las

metodologías ágiles se exige una gran capacidad de trabajo en grupo y buenas estrategias de comunicación, prescindiendo de una gestión centralizada y estricta para el manejo de los inconvenientes, riesgos y documentación. Pero también debido a la descentralización, los procesos de documentación, manejo de riesgos y control de cambios deben estar ligados a los equipos de trabajo y en el caso de la documentación al producto en sí, recurriendo a la capacidad del código de explicarse a sí mismo.

- Las etapas iniciales de las metodologías ágiles, deben ser ejecutadas de manera estricta y poniendo especial atención al esfuerzo que se llevará acabo para el proyecto, pues una mala estimación puede generar retrasos en el tiempo establecido para las entregas. También es importante conocer las habilidades y capacidades del equipo de trabajo, pues sin esta información es muy difícil calcular la duración y la cantidad de objetivos que se pueden cumplir para cada iteración.

- Lo que más se puede destacar de una metodología ágil está relacionado con el ambiente de trabajo que esta genera en los proyectos de desarrollo de software, pues su objetivo es mantener al equipo motivado, unido y en constante comunicación con el dueño de las necesidades para precisamente dar solución a lo que realmente requiere sin recurrir a intermediarios o medios de comunicación que podrían tener algún ruido o interferencia en lo que a las especificaciones del producto se refiere.

- Un buen conjunto de herramientas para apoyar el proceso de desarrollo de software puede significar un considerable ahorro de tiempo y dinero en los proyectos informáticos; si bien son importantes las herramientas para el análisis y diseño del sistema, son más decisivas las que intervienen en la parte de desarrollo, diseño de interfaz de usuario y pruebas, pues son las labores más largas y tediosas que implican gran capacidad de organización, pensamiento y habilidades de programación. Si se carece de alguna herramienta que automatice y sirva como estándar para la ejecución de estas actividades, entonces tomará mucho tiempo organizarse y se tendrá que hacer todo desde el principio para cada proyecto.

# GLOSARIO

**ARTEFACTO:** un artefacto en el contexto de desarrollo de software, es un subproducto o entregable físico que se produce al ejecutar las actividades del proyecto, pueden ser artefactos: documentos, prototipos, gráficos, etc.

**BACKLOG:** Se utiliza como término de la metodología SCRUM y consiste básicamente en una lista de requisitos que los usuarios tienen para el sistema que se va a desarrollar.

**BUG:** Un bug es un error o fallo que se presenta en un sistema.

**DESPLIEGUE:** Se refiere al montaje del software sobre alguna plataforma para que este pueda ser usado.

**ELICITACIÓN:** Obtención de información, generalmente se utiliza este término como elicitación de requisitos y se refiere a una etapa en la que se obtiene la información y los requisitos que el usuario desea que contenga el software a desarrollar.

**FRAMEWORK:** Un framework como término usado en el desarrollo de software, es un marco, estructura o esqueleto que se usa como base para hacer el trabajo. Propone unas métricas y un mapa a seguir para agilizar y estandarizar el proceso de fabricación del producto software.

**HEURÍSTICAS:** Asociado al pensamiento humano, es la capacidad de resolver problemas, inventar y descubrir, utilizando la creatividad y el pensamiento lateral o fuera de los patrones habituales.

**METODOLOGÍA INCREMENTAL:** Metodología en la que se repite varias veces las etapas de desarrollo de software, pero cada vez se va mejorando y agregando nueva funcionalidad hasta terminar el producto completo.

**METODOLOGÍA ITERATIVA:** Metodología en la que se repite varias veces las etapas de desarrollo de software. Al final de cada ciclo se genera un producto funcional completo mejorando al del ciclo anterior. El proceso termina al generar un producto satisfactorio

**OPEN SOURCE:** Software open source o de código abierto a grandes rasgos, es aquel que nos permite tener acceso al código fuente, modificarlo y distribuirlo libremente sin necesidad de estar atados o pagar por alguna licencia a quienes desarrollaron el producto.

**PRUEBAS DE INTEGRACIÓN:** Son pruebas que se ejecutan al unir módulos o funcionalidades que han sido desarrolladas por separado.

**PRUEBAS UNITARIAS:** Son pruebas que se ejecutan sobre un módulo, funcionalidad o porciones de código. A diferencia de las pruebas de integración, estas se hacen sobre una porción de código que se ha desarrollado conjuntamente.

**RELEASE:** Palabra muy utilizada para referirse una versión del software que ya es funcional.

**REQUISITO:** Un requisito es una condición necesaria para algo, es una necesidad que tiene que ser satisfecha para que el sistema funcione como debe hacerlo.

**SPRINT:** Término utilizado por la metodología SCRUM para referirse a una iteración, ciclo o intervalo de tiempo en el cual el equipo de trabajo se encarga de llevar a cabo un incremento funcional del producto que se está desarrollando de acuerdo a los requisitos propuestos para ese período.

**TESTER:** Persona o rol encargado de hacer pruebas sobre el sistema o software que se está desarrollando.

**TIMEBOX**: Iteración, ciclo o intervalo. Corresponde a una palabra clave utilizada en las metodologías Crystal y DSDM para referirse al período de tiempo durante el cual se hace un aumento funcional sobre el software que se está desarrollando.

# BIBLIOGRAFÍA


Schenone Marcelo Hernán, diseño de una metodología ágil de desarrollo de software. [En línea]. Argentina. *2004 http://materias.fi.uba.ar/7500/schenone-tesisdegradoingenieriainformatica.pdf* [Consultado en Septiembre del 2011]

Amaro Calderón, Sarah Dámaris, Valverde Rebaza, Jorge Carlos, metodologías ágiles. [En línea]. Perú. 2007. *http://es.scribd.com/doc/55710897/Metodologias-Agiles.pdf* [Consultado en Septiembre del 2011]

Carvajal Riola, Jose Carlos, metodologías ágiles, herramientas y modelo de desarrollo para aplicaciones Java EE como metodología empresarial. [En línea]. España. 2008 *http://upcommons.upc.edu/pfc/bitstream/2099.1/5608/1/50015.pdf* [Consultado en Septiembre del 2011]

Canós, José H, Letelier, Patricio, Penadés, Mª Carmen, metodologías ágiles en el desarrollo de software. [En línea]. España. *http://www.willydev.net/descargas/prev/TodoAgil.Pdf* [Consultado en Septiembre del 2011]

Rational Software White Paper TP026B, Rev 11/01. [En línea]. International Business Machines, Corp. 2001 *http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf* [Consultado en Septiembre del 2011]

Sandra Lorena Anaya, RUP vs XP. [En línea] *http://www.iered.org/archivos/Grupo_Vultur/Seminario/3-mecs/sesion3/RUP-Vs-XP.pdf* [Consultado en Septiembre del 2011]

Manifesto for Agile Software Development. [En línea]. 2001. *http://agilemanifesto.org/* [Consultado en Septiembre del 2011]

Coad Letter Process Editor, Tools and Agile Processes. [En línea]. *http://edn.embarcadero.com/article/29687* [Consultado en Septiembre del 2011]

Cockburn, Alistair, Surviving Object Oriented Projects, Addison-Wesley Object Technology Series, 1998.

Extreme Programming. [En línea]. 2009. *http://www.extremeprogramming.org* [Consultado en Septiembre del 2011]

El método SCRUM. [En línea]. 2007. *http://www.mastersoft.com.ar/MsWeb/otros_archivos/NotaScrumPCUsers.pdf* [Consultado en Septiembre del 2011]

Juan Palacio, Claudia Ruata, Scrum Manager: Proyectos - apuntes de formación. [En línea]. 2009. *http://www.slideshare.net/slimshadyx18/scrum-manager-proyecto-apuntes* [Consultado en Septiembre del 2011]

Raúl Jiménez Ortega, introducción a SCRUM. [En línea]. Asociación de Webmasters de Granada. *http://osl.ugr.es/talleres/SCRUM/Presentacion%20SCRUM.html#slide6* [Consultado en Septiembre del 2011]

Margarita Fernández Enrich, Crystal Methodologies. [En línea]. 2003. *www.dsic.upv.es/asignaturas/facultad/lsi/trabajos/282002.ppt* [Consultado en Septiembre del 2011]

Anova Ingeniería de Software. [En línea]. 2008 *http://www.anovanet.es/metodologia/metodologia.html* [Consultado en Septiembre del 2011]

Alistair Cockburn, The Crystal Methods, or How to make a methodology fit. [En Línea]. 2003 *http://agile2007.agilealliance.org/downloads/handouts/Cockburn_818.pdf* [Consultado en Septiembre del 2011]

Numan Muhammad, Chrystal methodologies a family of lightweight, agile software development methodologies. [En línea]. Pakistan. *http://es.scribd.com/doc/54257803/Crystal-Methodologies* [Consultado en Septiembre del 2011]

Steven Thomas, Software Development An Agile Comparison. [En línea]. 2011. *http://www.balagan.org.uk/work/agile_comparison.htm* [Consultado en Septiembre del 2011]

Scott W. Ambler, The Agile Unified Process (AUP) [En línea]. 2009 *http://www.ambysoft.com/unifiedprocess/agileUP.html* [Consultado en Septiembre del 2011]

Kent Beck, Scrum with XP and Beyond. [En línea]. 2007 *http://www.gbcacm.org/sites/www.gbcacm.org/files/slides/7A%20-%20Jeff%20on%20Scrum%20and%20XP.pdf* [Consultado en Septiembre del 2011]

The Eclipse Foundation, OpenUP [En línea]. 2011. *http://epf.eclipse.org/wikis/openup/* [Consultado en Septiembre del 2011]

Roberth G. Figueroa, Camilo J. Solís, Armando A. Cabrera, metodologías tradicionales vs. metodologías ágiles. [En línea]. *http://www.buenastareas.com/ensayos/Modelos-De-Procesos-De-Software/2600270.html* [Consultado en Septiembre del 2011]

Mayra Ofelia Hernández Martínez, Alicia del Coral Landa Valladares, Cesar Uriel Mojica Hernández, RUP. [En línea]. 2010. *http://www.slideshare.net/aliciadelcoral/exposicion-rup-5303096* [Consultado en Septiembre del 2011]

Daniel Sócola Escobar, Metodologías y ciclos de vida. [En línea]. *http://www.slideshare.net/vdaniel20/metodologas-y-ciclos-de-vida* [Consultado en Septiembre del 2011]

What is TortoiseCVS? [En línea]. TortoiseCVS. http://www.tortoisecvs.org/index.shtml [Consultado en Septiembre del 2011]

JUnit. [En línea]. JUnit.org. http://www.junit.org/ [Consultado en Septiembre del 2011]

Checkstyle. [En línea]. Sourceforge. http://checkstyle.sourceforge.net/ [Consultado en Septiembre del 2011]

Generate components quickly with AndroMDA. [En línea]. AndroMDA.org. http://www.andromda.org/ [Consultado en Septiembre del 2011]

Marc Clifton, J.Dunlap, What is DSDM?. [En línea]. 2003 http://www.codeproject.com/KB/architecture/dsdm.aspx [Consultado en Octubre del 2011]

Best Practices in Scrum Project Management and XP Agile Software Development [En línea] Object Mentor, Inc. 2004. http://www.objectmentor.com/resources/articles/Primavera.pdf [Consultado en Octubre de 2011]

David Putman, Workshare Technology and eXtreme Programming (XP). [En línea]. Workshare Technology. http://www.objectmentor.com/resources/articles/workshare.zip [Consultado en Octubre de 2011]

Improving Outcomes through Agile Project Management [En línea] General Dynamics United Kingdom Limited. 2010. *http://www.dsdm.org/wp-content/uploads/2011/02/Improving-Outcomes-Through-Agile-Project-Management.pdf* [Consultado en Octubre de 2011]

NORMA TÉCNICA COLOMBIANA NTC 1486 (Cuarta actualización) [En línea]. http://www.eafit.edu.co/practicas/estudiantes-practica/Documents/norma_tecnica_colombiana_ntc1486.pdf [Consultado en Octubre de 2011]

March Hare Pty Ltd, & CVSNT, TortoiseCVS, WinCVS Projects [En línea]. 2008 http://march-hare.com/cvsnt/features/cvsnt/ [Consultado en Octubre de 2011]