Story Points y su Importancia en la Estimación de Proyectos Bajo el Enfoque Ágil

Importance of Story Points in the Estimation of Projects Under the Agile Approach

Adriana María Iglesias Solano
aiglesias3@unisimonbolivar.edu.co
Universidad Simón Bolívar
Barranquilla – Atlántico

Palabras clave:
Story point, User Story, estimación ágil, métricas, Story, Scrum, Xp.

Resumen
Un proyecto que es abordado bajo el enfoque de desarrollo ágil de ser capaz de adaptarse a los cambios, especialmente a los requerimientos del cliente, el objetivo a la hora de encarar este tipo de proyectos es minimizar el tiempo y el costo de su realización, pero manteniendo la calidad del software y los procesos. Es por esto que el saber estimar y planificar para proyectos de desarrollo ágil es una labor trascendental y fundamental, porque no se debe perder el objetivo de este enfoque: la agilidad, centrándose en el uso de métricas que se ajusten y no se conviertan en el talón de Aquiles del equipo. En este artículo, presento algunos principios y prácticas introductorias para aprender a estimar un proyecto ágil bajo la métrica de los Story Points.

Key-words:
Story point, User Stories, agile estimation, metric, Story, Scrum, Xp

Abstract
A project that is boarded under the approach of agile development of being able to adapt to the changes, especially to the requirements of the client, the objective at the time of facing this type of projects is to diminish the time and the cost of its accomplishment, but maintaining the quality of software and the processes. It is by that the knowledge to consider and to plan for projects of agile development is a transcendental and fundamental work, because the objective of this approach is not due to lose: the agility, concentrating in the use of metric that adjusts and does not become the heel of Aquilles of the equipment. In this article, I present/display some principles and introductory practices to learn to consider an agile project under the metric one of the Story Points.

I. INTRODUCCIÓN

El mercado actual en la industria del desarrollo de software está caracterizado por la tendencia hacia el desarrollo rápido de aplicaciones, es por este motivo que la ventaja competitiva de dichas empresas desarrolladoras radica en aumentar la productividad y adaptarse a entornos cada vez más cambiantes, teniendo como objetivo principal crear aplicaciones acordes a las necesidades de los clientes, con la tecnología que ellos requieren y en el menor tiempo posible. Ante la situación planteada anteriormente, las llamadas metodologías tradicionales o pesadas han suscitado cierta cantidad de inconvenientes de adaptabilidad ante estos entornos cambiantes y cortos períodos de desarrollo, esto debido en gran parte a la planificación previa y el establecimiento de objetivos y requerimientos al inicio del ciclo de vida de desarrollo de software. Gracias a esto las metodologías ágiles de desarrollo han tomado fuerza en un intento de mejorar las falencias ocasionadas como consecuencia del uso de las metodologías tradicionales, a pesar de que este término fue acuñado a mediados de los 90's es en la actualidad en donde su aplicación es cada vez más común. Según Kent Beck, uno de los padres del modelo ágil, *"cada cosa en software cambia. Los requisitos cambian, el diseño cambia, el negocio cambia, la tecnología cambia, el equipo cambia y los miembros del equipo cambian. El problema no es el cambio, porque el cambio, inevitablemente, va a ocurrir. El problema, realmente, es nuestra inhabilidad para hacer frente a los cambios"*.

Estos cambios a los que hace referencia Kent Beck surgen como consecuencia de cuatro aspectos relevantes: a) los requerimientos rara vez están totalmente definidos y claros al inicio del proyecto. b) la incorporación de nuevas tecnologías y/o herramientas al proyecto desconocidas, hace más complicada la labor de establecer a priori las estrategias de trabajo. c) Es normal y común que los requerimientos sufran cambios durante la ejecución de las fases de desarrollo. d) al ver una versión de la aplicación a algunos clientes les da una idea más clara de lo realmente quieren.

Éstas metodologías de desarrollo ágil aplican un ciclo de desarrollo incremental e iterativo, permitiendo así hacer entregas en partes pequeñas, funcionales y utilizables; cada iteración puede ser considerada como un pequeño proyecto en el que las actividades de: requerimientos, diseño, pruebas, e implementación son un subconjunto del sistema general; este proceso es repetido en varias iteraciones generando un incremento en cada iteración hasta que se complete el sistema.

Es por esto que los procesos de estimación y planificación en un proyecto ágil son algo diferentes a los que se aplican en un proyecto bajo un enfoque tradicional. En un proyecto tradicional, el proceso de estimación es por etapas; se planifica el desarrollo y luego se ejecuta, Pero cuando surgen contratiempos y atrasos comenzaran las complicaciones

II. MODELO ÁGIL DE DESARROLLO DE SOFTWARE

Las metodologías de desarrollo ágil a pesar haber sido formuladas a mediados de los años 90, es en la actualidad en donde su uso se ha proliferado, estas metodologías nacieron con el afán de reemplazar a las tradicionales o también denominadas metodologías pesadas de desarrollo de software. Fue entonces como en el año 2001 algunas de las personas afines con los principios de estas metodologías se agruparon formando lo que conocemos hoy como Alianza Ágil, que promueve el desarrollo ágil de aplicaciones. Es de esta forma como actualmente cada vez mas empresas de desarrollo le apuestan al uso de estas metodologías para la construcción de sus productos de software y ha sido así como su popularidad ha ido cada vez más en aumento.

Pero, ¿Por qué utilizar metodologías ágiles?, las metodologías de desarrollo de software tradicionales o pesadas han demostrado presentar ciertos problemas, entre ellos se pueden citar los siguientes:

- La ingeniería de requisitos es una etapa inmersa en el análisis, la formulación de estos requisitos deben reflejar exactamente lo que el cliente desea que realice su programa de aplicación, tratando de evitar que se produzcan cambios de este listado de requisitos establecido, debido que

a medida en como avance el proyecto por cada una de sus etapas de desarrollo será más costosa la realización de dichos cambios o corrección errores. Partiendo de lo anterior se puede afirmar que se hace necesario que la especificación de requisitos y las entrevistas con el cliente sean desarrolladas iterativamente y con cierta frecuencia, debido a que en gran parte de las veces el cliente no conoce sus necesidades con la profundidad suficiente como para definirlas exactamente al inicio del proyecto.

- Otra dificultad de las metodologías tradicionales es lo lento que puede ser el proceso de desarrollo, debido a que es difícil la mayoría de las ocasiones que el equipo de desarrollo entienda en su totalidad un sistema complejo

- El equipo debe realizar un enorme esfuerzo en la etapa de planeación, debido a que como se mencionó anteriormente la corrección de errores en las metodologías tradicionales son muy costosos y estos redundaran en el presupuesto del proyecto y deben ser asumidos por alguna de las partes o por ambos.

- En algunas ocasiones y debido en gran parte a los factores ya mencionados el resultado final no es exactamente lo que quiere el cliente o no tiene la calidad esperada.

- Los retrasos en la entrega de productos son el pan de cada día o lo mismo que la incorrecta ejecución de las últimas fases del ciclo de vida

En contraposición a todo lo anterior, suscitaron las metodologías ágiles como alternativa de solución a las dificultades esbozadas del uso de metodologías tradicionales y se ajustan cuando los requisitos están en expuestos a constante cambio y en este contexto presenta las siguientes ventajas:

- Se realizan entregas continuas del producto en cortos lapsos de tiempo, lo que permite al cliente disfrutar de las funcionalidades del software a medida que se va construyendo.
- Los riesgos y dificultades se reparten a lo largo del proceso de desarrollo, lo que permite realizar retroalimentaciones constantes y establecer controles a los riesgos.
- La capacidad de respuesta a los cambios es mayor que con la metodología tradicional, ya que estos son percibidos como oportunidades de mejora del software en construcción.
- Mejora continua de los procesos y el equipo de desarrollo. El trabajo conjunto entre el cliente y el equipo de desarrollo es de vital importancia para el buen manejo del proceso y la aplicación de buenas prácticas, gracias a este fuerte vínculo se minimizan los malentendidos o inconsistencias a la hora de hacer la labor de elicitación.

III. VALORES, PRINCIPIOS Y PRÁCTICAS DEL MANIFIESTO ÁGIL

El desarrollo de software a través de la aplicación de metodologías ágiles no especifica unos procesos o métodos a seguir, por el contrario este establece cuatro (4) valores y doce (12) principios.

1. Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas.
2. Desarrollar software que funcione por encima de una completa documentación.
3. La colaboración con el cliente por encima de la negociación contractual.
4. Responder a los cambios más que seguir estrictamente un plan.

Para cumplir estos valores se siguen doce principios:

1. La prioridad es satisfacer al cliente mediantes tempranas y continuas entregas de software que le aporten valor.
2. Dar la bienvenida a los cambios de requisitos. Se capturan los cambios para que el cliente tenga una ventaja competitiva.
3. Liberar software que funcione frecuentemente, desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.
4. Los miembros del negocio y los desarrolladores deben trabajar juntos diariamente a lo largo del proyecto.
5. Construir el proyecto en torno a individuos motivados. Darles el entorno y apoyo que necesiten y confiar a en ellos para conseguir finalizar el trabajo.

6. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo
7. El software que funciona es la principal medida de progreso.
8. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.
9. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
10. La simplicidad es esencial.
11. Las mejores arquitecturas, requisitos y diseños surgen de los equipos que se auto organizan.
12. En intervalos regulares el equipo debe reflexionar sobre cómo ser más efectivo y según estas reflexiones ajustar su comportamiento.

Estos principios marcan el ciclo de vida de un desarrollo ágil, así como las prácticas y procesos a utilizar.

- Planificación
- Programación en pareja
- Integración continua
- Refactorización.
- Desarrollo dirigido por pruebas

IV. PRINCIPALES ENFOQUES ÁGILES

Existen diversos enfoques, modelos y metodologías que enmarca la filosofía ágil en este caso bordaremos dos (2) de ellos como lo son: eXtreme Programming y Scrum.

Extreme Programming (XP)

Tiene su aparición en el año 1999, el objetivo de XP es potenciar las relaciones interpersonales como clave del éxito en proyectos software que se desarrollan en entornos con requerimientos muy cambiantes e imprecisos. XP está fundamentado en una serie de valores que lo rigen, los cuales son:

- Comunicación fluida entre todos los participantes.
- Simplicidad en las soluciones implementadas.
- Continúa retroalimentación entre el cliente y el equipo de desarrollo.
- Coraje para enfrentar los cambios. Detrás de este valor encontramos el lema "si funciona, mejóralo".

SCRUM

Es un marco de referencia utilizado para el desarrollo ágil de aplicaciones de software, sus principales características son:

- División de la organización en pequeños equipos, interdisciplinarios y organizados.
- División del trabajo. Esta se debe organizar en una lista de entregables pequeños y concretos, esta lista debe ser organizada según la prioridad y la estimación de cada elemento.
- División del tiempo. Esto se realiza en iteraciones cortas de longitud fija, normalmente con una duración entre una (1) a cuatro (4) semanas, con código potencialmente entregable y funcional después de cada iteración.
- Actualiza Las Prioridades. De la mano con el cliente, basada en los conocimientos adquiridos mediante la verificación del entregable después de cada iteración

User Stories

Los User Stories son el elemento base que utiliza SCRUM para describir las características que el usuario espera que tenga el software que se va a desarrollar. Por lo tanto, pueden incorporar tanto cuestiones relacionadas con las funciones del sistema como con cualquier otro aspecto del mismo (restricciones, rendimiento, etc.).

Las historias de usuario se presentan desde la perspectiva del usuario. Así, no se describen utilizando una terminología técnica sino que se escriben utilizando un lenguaje cercano al dominio de la aplicación que se está desarrollando de forma que sea comprensible por los clientes y por los desarrolladores.

Actores y Roles

Los actores que intervienen dentro del proceso son de forma general los siguientes:

- Propietario del producto ó Product Owner,
- Maestro de scrum ó Scrum Master,
- Equipo de desarrollo ó Scrum Team y,

- Cliente o usuario

V. ESTIMACIÓN TRADICIONAL

Entre las medidas de estimación tradicionales se pueden enunciar los muy conocidos Puntos de función y la de Líneas de código que de ahora en adelante denominaremos PF y LOC (Line Of Code) respectivamente.

Antes de continuar definiendo LOC y PF, mejor recordemos ¿Qué es una estimación y por qué de su importancia en los proyectos de desarrollo de software?, la estimación es una medida una métrica que nos ayuda a calcular el esfuerzo o tiempo que requiere para llevarse a cabo un proyecto, cabe mencionar que dichos cálculos son aproximaciones al esfuerzo real debido a que es muy difícil por la misma naturaleza de los proyectos de desarrollo de software realizar un cálculo exacto.

Ahora sí, iniciemos abordando la medida de estimación de LOC, su propósito es definir el tiempo y el costo del proyecto con base a la cantidad de líneas de código que se tienen que escribir durante el desarrollo, pudiéndose determinar así, el costo por línea y cuantas líneas de código se desarrollan por mes. Pero LOC tiene ciertas dificultades entre las cuales se podrían nombrar las siguientes:

- El número de LOC dependerá en gran parte de la habilidad y destreza del programador que lo esté realizando.
- Existe una relación inversa entre el nivel del lenguaje (lenguaje de alto nivel o lenguaje ensamblador) y la producción de trabajo.
- El número real de LOC no se sabrá hasta que el desarrollo del software esté casi terminado.
- No hay un método aceptado para contar líneas de código, aunque existen diversas aplicaciones que contribuyen a la realización de esta labor, entre las cuales se podrían nombrar: CodeCount, CCCC, K-LOC Calculator, EZMetrix, Code Analyzer, solo por mencionar algunos.

Por otra parte, los PF son utilizados para la estimación en función de los atributos del sistema a desarrollar. Según Pressman, “los PF se obtienen a partir de una relación empírica basada en medidas contables del dominio de información de software y la evaluación de la complejidad del software”. Para realizar el cálculo basado en PF se deben tener en cuenta las siguientes propiedades:

- Número de entradas de usuario.
- Número de salidas del usuario
- Número de consultas de los usuarios
- Número de archivos
- Número de interfaces externas

Una vez que estas propiedades han sido colocadas en la tabla se elige un peso, el cual representará el grado de complejidad que se prevé para cada una de las opciones. El cálculo de los PF se realizará entonces basado en la siguiente fórmula:

$$PF=\text{Cuenta Total} * [0.65+0.01*\text{SUM}(f1)]$$

donde la variable función SUM (f1), será el resultado de la sumatoria de aplicación de las siguientes preguntas asociadas a su respectiva puntuación en el rango de 0-5 (0 no influyente y 5 esencial)

1. ¿Requiere el sistema copias de seguridad y de recuperación fiables?

2. ¿Se requieren comunicaciones de datos?

3. ¿Existen funciones de procesamiento distribuido?

4. ¿Es crítico el rendimiento?

5. ¿Será ejecuta el programa en un sistema operativo existente y fuertemente utilizado?

6. ¿Requiere el sistema entrada de datos interactiva?

7. ¿Requiere la entrada de datos interactiva que se utilicen varias pantallas o varias operaciones?

8. ¿Se utilizan los archivos maestros de forma interactiva?

9. ¿Son complejas las entradas, las salidas y/o las peticiones?

10. ¿Es complejo el procesamiento interno?

11. ¿Se ha diseñado el código para ser reutilizable?

12. ¿Están incluidas en el diseño la conversión y la instalación?

13. ¿Se ha diseñado el sistema para soportar diferentes instalaciones en diferentes organizaciones?

14. ¿Se ha diseñado la aplicación para facilitar los cambios y para ser fácilmente utilizada por el usuario?

Los PF los podemos utilizar para calcular:

- Productividad = PF/Personas-mes
- Calidad = Errores/PF
- Costo = Pesos/PF
- Documentación = Páginas /PF

VI. ESTIMACIÓN EN PROYECTOS ÁGILES

En los numerales II, III y IV hemos conocido un poco sobre el modelo ágil de desarrollo de software, sus principios y valores y analizamos las características de dos enfoques ágiles muy populares entre los desarrolladores actuales como lo son:
Scrum y XP. En el numeral V tratamos sobre la estimación en proyectos tradicionales, analizando los PF Y LOC, ahora la inquietud a resolver seria si estas dos estimaciones son de aplicación pertinente en proyectos ágiles.

Tal y como ya se mencionó sabemos que las metodologías ágiles deben ser adaptables a las necesidades cambiantes en cualquier momento durante la vida del proyecto, a diferencia del enfoque tradicional que tratar de definir todos los requisitos en el inicio del proyecto y luego ampliar los esfuerzos para controlar los cambios a los requisitos. Es por esto que las metodologías Ágiles trabajan bajo el principio de entrega de una salida de trabajo, reunir otros requisitos y evolucionar el producto.

Partiendo de lo visto anteriormente veremos las propuestas para estimar en proyectos ágiles, centrémonos en los ***Story points.***

A. STORY POINTS

Se podría afirmar que: “Los Story points son una medida de complejidad y esfuerzo, algunos equipos optan por tomarlos como medida complejidad; pero estos también se pueden orientar para estimar el esfuerzo requerido para desarrollar un producto, incluyendo los riesgos y la incertidumbre. Definamos entonces un Story Point a partir de la siguiente ecuación:

Story Point = Esfuerzo + Complejidad + Riesgo

Analicemos el siguiente contexto: Supongamos usted se debe dirigir a su oficina,” ¿Qué medio de transporte usaría para llegar?”, puede que usted decida tomar un taxi o un autobús, dependiendo del tráfico y "¿Cuánto tiempo se tardaría en ir de casa a la oficina?", multitud de otros factores variables influirán de una forma u otra en su resultado final que debe ser en este caso llegar a la oficina. Por otro lado, si ahora nos preguntamos "¿Cuál es la distancia entre la casa y la oficina?", La respuesta a este interrogante será una constante. Lo cual nos lleva a afirmar que “No importa cuál sea el medio de transporte que se use o el tráfico, la distancia entre la casa y la oficina sigue siendo la misma. Bueno, ahora pensemos en los Story Points, estos son como la estimación de la distancia. Ellos siguen siendo los mismos, sin importar quién está trabajando en los Story. El tiempo puede variar en función de muchos factores: La persona que trabaja en el Story, el tiempo de reunión, tiempo perdido, todo contribuye al tiempo real invertido, pero el tamaño sigue siendo el mismo.

Los Story Points son relativos, es decir, supongamos que durante el desarrollo de un proyecto se elige un User Stories como referencia y se le asigna un valor de 1 punto. A continuación, se debe medir cada Story en relación con éste, de tal forma que a medida en que aumenta la complejidad así aumentará el valor del punto. Los Story Points no intentan asignar la duración de cada Story, sino que permiten una comparación entre las tareas.

Los Story deben ser:

- Comprensibles para los clientes y desarrolladores.
- Comprobables
- Del tamaño justo. Se recomienda que sean pequeños como para que el programador pueda crear varios en una iteración

B. ESTIMACIÓN EN STORY POINTS

Los Story Points indican el tamaño y la complejidad dado un User Story con relación a otro story que son parte del proyecto. Determinar el número de Story points en cada Story es subjetivo. Los Story points permiten estimar esfuerzo sin tratar de estimar cuanto tiempo tomará.

Ahora si nos cuestionamos sobre ¿Cuánto tiempo se tarda en desarrollar un software?, la respuesta sería: se suman los Story Points y luego dividimos entre la velocidad, y eso nos daría el tiempo:

Velocidad del equipo = Story Points / Iteración

Pero qué tan grande es un punto de la historia? ¿Y cómo puedo saber lo que la velocidad es?

Si estamos trabajando bajo el enfoque de Scrum, las estimaciones de los Story Points son realizadas por el equipo que realiza el desarrollo. Mirándolo desde esta perspectiva, son ellos los que definen un Story Point.

Algunos desarrolladores estiman los Story points basados en la experiencia y conocimientos, a partir de ahí, hacen un estimativo del desgaste que empleará la realización de esa tarea; es así como son asignados los niveles de los Story Points. Esta es la forma más sencilla para estimar Story referenciándose a un Story similar producto de la experiencia, en este caso, el Story probablemente tendrá el mismo valor que un Story similar.

Otros lo estiman, leyendo la lista de tareas a realizar, toman lo que parece ser lo más sencillo, basados en la experiencia, y le asignan el valor de 2. Y a partir de allí, todo lo demás se estima en relación con esa tarea.

Pero si analizamos lo anterior, nos damos cuenta que estimar de esta forma puede llegar a ser muy impreciso. Más formalmente, los Story Points se calculan por lo general en la escala de Cohn, denominado de esta forma por Mike Cohn, esta es la escala de estimación:

0, 1, 2, 3, 5, 8, 13, 20, 40, 100.

Ahora surge una duda, ¿Cómo calculamos la velocidad?, en el primer sprint, el equipo acepta el trabajo basados en la experiencia y el debate. Después de cada Sprint, se mide la velocidad. Después de dos o tres sprints, la velocidad media de medición pueden ser utilizados para predecir la velocidad en el futuro, y por lo tanto la fecha de finalización

C. Técnicas de estimación - Planning Poker

Planning Poker, es una técnica de estimación para proyectos ágiles propuesta en un inicio por Grenning en el año 2002 y popularizada por Cohn, la cual permite hacer una estimación inicial del proyecto rápida. Esta estimación se realiza de manera consensuada con base en los requisitos.

El proceso de Planning poker inicia con una reunión entre los miembros del equipo, se plantea un Story, se analiza y utilizando las cartas cada miembro muestra cuanto cree que sea el valor que debe asignarse. Como número resultante se puede tomar la más alta, la media, la más baja, o cualquier otro método; recuerde que el valor asignado debe ser calculado con un método consensuado por todos los miembros del equipo.

Una buen método que puede facilitar la tarea de consenso es preguntar al equipo el por qué de determinados valores, después de escuchar las razones se propone volver a votar.

Para estimar por medio de esta técnica necesitaremos:

- Una lista de características, basada en los User Stories la cual describirá el software a desarrollar.
- Una baraja de cartas, estas cartas se encuentran numeradas, cabe resaltar que existen dos tipos de barajas. Una contiene cartas con la secuencia de Fibonacci incluyendo un cero:

0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89.

Otros contienen progresiones similares. La razón de utilizar la secuencia de Fibonacci es reflejar el incierto inherente en la estimación. Sin embargo, la baraja usada comúnmente es la secuencia:

0, ½, 1, 2, 3, 5, 8, 13, 20, 40, 100,

La baraja también trae incluida opcionalmente, cartas con los símbolos de "?", que significa inseguridad o duda y una taza de café, que hace referencia a la necesidad de un break o descanso.

La carta con el cero (0) significa que este story ya ha sido realizado.

D. HERRAMIENTAS DE SOFTWARE RECOMENDADAS

Pivotal Tracker (http://www.pivotaltracker.com/)
Software para la gestión ágil de proyectos, disponible en-linea de forma gratuita, trabaja bajo la filosofía de cloud computing. En la experiencia de uso que he tenido con la aplicación me hace recomendarla para la gestión de los user story y la asignación de los Story points, ya que la aplicación es muy usable e intuitiva, sitúa cuatro áreas de en una dashboard.

Plannning Poker (www.planningpoker.com)
En el item anterior conocimos sobre esta técnica de estimación, vimos su modo de operación. Con este sitio web podemos gestionar el juego de estimación del Planning Poker, es una herramienta libre y también trabaja bajo la filosofía de cloud computing. Es de uso sencillo y muy seguro, maneja sesiones de usuario entre otras características.

VII. CONCLUSION

Después de finalizado el estudio de la temática se puede concluir que:

La estimación y planificación ágil permiten a través de sus técnicas y métricas conocer cuál podría ser la fecha estimada de finalización del proyecto, y mejor aún saber en qué iteración estará lista determinada funcionalidad del producto a desarrollar.

Las medidas de tamaño en el enfoque tradicional y en el ágil son diferentes. Mientras que el primero tenemos los puntos de función y las líneas de código, en el segundo enfoque tenemos los Story points y los Ideal Days

Las estimaciones relativas son de mejor comportamiento que las absolutas.

Es más recomendable estimar en Story Point y el uso de la velocidad para calcular el tiempo tal como el ejemplo planteado del medio de transporte de la casa a la oficina. Lo cual nos ayuda a plantear que la distancia dividida por su velocidad le da el tiempo necesario. Buena fórmula, para calcular el tiempo que estimamos durará nuestro proyecto, ¿cierto?

Para obtener la velocidad inicial de nuestra primera iteración, estimamos que los story y los días hombre ideal. Digamos que el story es de 2 puntos y 4 días hombre ideal, entonces sabemos que el equipo puede manejar un medio punto del story por día hombre ideal para días de calendario. El factor de enfoque suele ser entre 50% y 70% dependiendo de la cantidad de apoyo y las interrupciones. Si el factor de atención es de 50% que puede manejar 4 story points en un día natural (1 / 2 * 50%). Si el equipo conformado por cinco personas y la iteración es de 14 días, tenemos entonces 70 días en la iteración. 70 * 4.1 da para que seamos capaces de llevar story points en la iteración. Por último tenemos la velocidad inicial estimada.

REFERENCIAS

[1] Boehm, B.: A View of 20th and 21st Century Software Engineering. In: 28th international conference on Software engineering, pp. 12--29. Shanghai, China (2006)

[2] Blum: Software development a holistic View, Oxford Press.466-467

[3] Canós, J.H., Letelier, P., Penadés, M.C., "Metodología Ágiles en el Desarrollo Software". VIII Jornadas de Ingeniería de Software y Bases de Datos, (2003).

[4] Robinson, H. Sharp, L., "The social side of technical practices in eXtreme Programming and Agile Processes in Software Engineering", Lecture

Notes in Computer Science. Berlin: Springer Verlag, (2005), pags 139-147.
[5] www.agilealliance.org
[6] Cao, L., Ramesh, B.: Agile Requirements Engineering Practices: An Empirical Study. In: IEEE Computer Society Press Los Alamitos, CA, vol 25, pp-60-67 (2008)
[7] Palacio, Juan.: Flexibilidad con Scrum: Principios de diseño e implantación de campos de Scrum. SafeCreative. (2010)
[8] Alarcón, Pedro P. "Especificación de un modelo de operaciones aplicable a procesos de desarrollo y operación de sistemas con software". PhD thesis, Facultad de Informática. Universidad Politécnica de Madrid. (2008).
[9] Rodriguez, P., Yagüe, A., Alarcón, P.P., Garbajosa, J., "Metodologías ágiles desde la perspectiva de la especificación de requisitos funcionales y no funcionales". 13th Conference on Software Engineering and Databases (2008)
[10] Kniberg, H., Skarin, M.: Kanban Vs Scrum. InfoQ. (2010)
[11] Beck, K., et.al, Manifesto for Agile Software Development, http://agilemanifesto.org/
[12] Scrum Alliance, http://www.scrumaliance.org
[13] MendesCalo, K., Estevez, E., Fillottrani, P, "Un Framework para Evaluación de Metodologías Ágiles". (2008)
[14] Kniberg, H., "Scrum and XP from de trenches". http://infoq.com/minibooks/ scrum-xpfrom-the-trenches. (2007)
[15] Iacovelli, A., Souveyet, C., Framework for Agile Methods Classification, Workshop on Model Driven Informat. Systems Eng.: Enterprise, User and System Models (2008).
[16] http://alistair.cockburn.us/
[17] http://jeffsutherland.com
[18]www.nebulon.com/fdd y también www.featu redrivendevelopment.com
[19]ww.nebulon.com/articles/fdd/fddimplementatio ns.html
[20]Agile and Plan-Driven Methods Oil and Water? www.agilealliance.org/articles/reviews/Boehm1/ articles/agileAndPlanDrivenMethods.pdf
[21]Get Ready for Agile Methods, with Care www2.umassd.edu/SWPI/xp/papers/r1064.pdf
[22]Jørgensen, Magne. 2004. A Review of Studies on Expert Estimation of Software Development Effort.
[23]Agile Estimating and Planning. Mike Cohn. Prentice Hall, 2005
[24]Agile Software Development with Scrum. Ken Schwaber. Prentice Hall, 2001
[25]The art of Agile Development. James Shore and Shane Warden. Prentice Hall, 2007
[26]User Stories Applied: For Agile Software Development. Mike Cohn. Addison-Wesley Professional, 2004
[27]http://www.mountaingoatsoftware.com/system/ presentation/file/95/Cohn_SDWest2009_AEP.pd
[28]www.planningpoker.com