INSTITUTO TECNOLÓGICO DE COSTA RICA
ESCUELA DE INGENIERÍA EN COMPUTACIÓN
PROGRAMA DE MAESTRÍA EN COMPUTACIÓN

Días ociosos: modelo para medir la productividad durante todo el ciclo de vida del desarrollo de un software utilizando la ideología Agile

Tesis para optar por el grado de Magíster Scientiae en Computación, con énfasis en Ciencias de la Computación

Miguel Oviedo Bonilla

Tutora: Marisela Rodriguez

Cartago, Costa Rica

Enero 2015

Días ociosos: modelo para medir la productividad durante todo el ciclo de vida del desarrollo de un software utilizando la ideología Agile

Resumen

Cuántas veces nos preguntamos si nuestro equipo de desarrollo de software es eficiente o tan siquiera si es bueno para desarrollar software. Muchos grupos de desarrollo están utilizando la ideología Agile para mejorar sus procesos y ser más productivos. El problema es que la ideología ayuda a mejorar los procesos, pero no indica un modelo que se pueda seguir para medir la productividad del trabajo, por lo que no se puede saber si al implementar Agile, realmente surge alguna mejoría. Es muy difícil saber si un equipo de trabajo es productivo al producir software, ya que hay muchas variables que tomar en cuenta y muchas personas involucradas. En la presente tesis se define un modelo para medir la productividad de los grupos de desarrollo que utilizan Agile, específicamente cuando implementan Kanban. El modelo toma en cuenta todos los aspectos del ciclo de desarrollo de software utilizando Agile, desde las diferentes etapas del proceso hasta las diferentes personas que participan en el mismo. El modelo utiliza el concepto de "días ociosos" para medir la productividad, nos indica cuántos días el equipo de trabajo estuvo realmente sin trabajar o si tomó más días realizar una tarea porque se tuvo que rehacer el trabajo. Además, ayuda a saber dónde se encuentran los problemas del equipo. Todo esto va a facilitar la identificación de puntos a mejorar dentro del proceso de desarrollo y dentro del equipo de desarrollo. También da un punto de referencia para saber si se está haciendo el mejor uso de los recursos todo el tiempo y hacer comparaciones entre proyectos y equipos.

TEC | Tecnológico de Costa Rica
Programa de Maestría en Computación
Teléfono: 2550-2402

ACTA DE APROBACION DE TESIS

Con fundamento en lo que establecen los **Artículos 22-24-25** del "Manual de Normas y Procedimientos para optar por el título de "MAGÍSTER SCIENTIAE EN COMPUTACION", el Tribunal Examinador de Tesis (TET), nombrado con el propósito de evaluar la tesis de grado.

"Días ociosos: modelo para medir la productividad durante todo el ciclo de vida del desarrollo de un software utilizando la ideología Agile"

Habiendo analizado el resultado general del trabajo presentado por el estudiante:

Primer Apellido	Segundo Apellido	Nombre	No. de carné
Oviedo	Bonilla	Miguel	201174082

Emite el siguiente dictamen:

⊗ APROBADO El TET, considerando que el trabajo realizado por el estudiante es SOBRESALIENTE, le otorga la siguiente MENCION HONORIFICA: ⊗ CUM LAUDE ○ MAGNA CUM LAUDE ○ SUMMA CUM LAUDE	○ REPROBADO ○ SE RECOMIENDA ○ NO SE RECOMIENDA Brindarle una nueva oportunidad para la DEFENSA PUBLICA de su Tesis NUEVA FECHA: _____

Dando fe de lo aquí expuesto firmamos (*IDEM*: HOJAS DE APROBACION DE TESIS)

M.Sc. Marisela Rodríguez Aráuz
Profesor Asesor

M.Sc. Ignacio Trejos Zelaya
Profesor Lector

Master Luis Rodríguez Rodríguez
Profesor Externo

Dr. Roberto Cortés Morales
Coordinador del Programa
de Maestría en Computación

TEC | Tecnológico de Costa Rica
Maestría en Computación

01 de octubre de 2014

Sello

FT-07-MC

Dedicatoria

A mi madre, porque sin su esfuerzo y dedicación no tendría la educación que me permitió realizar todos los trabajos necesarios para elaborar esta tesis.

A mi esposa, por apoyarme a que realizara esta tesis inclusive durante el tiempo que estuvimos planeando la boda. Fueron tiempos difíciles, pero sin su apoyo no lo hubiera logrado.

Agradecimiento

A Dios.

Por la sabiduría y la fuerza de voluntad que siempre me acompañaron durante la realización de esta tesis.

A mi tutora de tesis Marisela Rodriguez.

Por la ayuda y guía brindada durante la realización de la tesis.

A los lectores Luis Rodriguez y Mónica Solera.

Por su tiempo invertido en la lectura y asesoría.

A mi equipo de trabajo.

Por ayudarme a realizar todas las pruebas y por los comentarios que ayudaron a completar la documentación necesaria.

Índice general

Índice de figuras

Índice de cuadros

1. Planteamiento del problema

Cada vez que se va a realizar una tarea, se quiere tener el mejor recurso humano y la tecnología más adecuada para realizarla lo mejor que se pueda y ser así totalmente productivo, o sea, utilizar al máximo todos los recursos asignados al proyecto y que no haya ningún desperdicio. La Real Academia Española, en su Diccionario de la Lengua Española, define la productividad como "la relación entre lo producido y los medios empleados, tales como mano de obra, materiales, energía, etc.". Es decir el costo del recurso invertido comparado contra el tiempo que se duró en realizar la tarea. Se puede decir que una buena productividad es cuando los recursos invertidos en hacer la tarea fueron utilizados en su totalidad o el 100% del tiempo que se duró. Esto es fácil de calcular para tareas sencillas o cuando se está trabajando en una sola, pero cuando se trata de algo tan complejo como el desarrollo de software, no es trivial hacer la comparación de los recursos contra el tiempo. El desarrollo de una aplicación puede durar mucho tiempo, inclusive años y hay muchos recursos que se invierten en las diferentes etapas del proceso. Para obtener lo que costó el proyecto se puede calcular el gasto total en salarios, así como la depreciación del equipo que se haya utilizado y facturas de los gastos que se generaron, pero todo esto simplemente dejaría un monto que no indica si se hizo bien, o sea, si se utilizaron al máximo los recursos o si hubo momentos en los que los recursos estuvieron ociosos o no utilizados.

Como ejemplo, se puede poner en perspectiva el traslado de una persona de un punto a otro. Se puede realizar en bicicleta, en automóvil o en transporte público. Si se hace en bicicleta se va a tener un bajo costo, sólo el deterioro de las llantas o de la cadena, por lo que puede parecer productivo, pero si se calcula el tiempo que se dura y se le pone un valor

a los minutos que se toma en realizar el traslado, se va a calcular que el costo sube, ya que el tiempo va a ser mucho mayor que en automóvil o transporte público. Si se realiza en transporte público, el costo y el tiempo serán menores que los de la bicicleta, pero el riesgo es mayor, ya que puede suceder que el transporte no pase a la hora esperada, o que tenga algún tipo de desperfecto en su camino, lo que elevaría el costo. Por último, se tiene el automóvil, el cual se utilizaría en el horario más conveniente, pero el costo sería mayor, ya que no se comparte el costo de la devaluación del automotor ni el costo de la gasolina con otros pasajeros, por lo que es mayor. Tomando en cuenta todas estas variables, saber cuál medio de transporte es más productivo a la hora de realizar el traslado se vuelve una tarea más complicada, no es suficiente comparar el costo y la duración.

En el desarrollo de software se da exactamente el mismo problema, según la ruta que se utilice para completarlo, se puede incurrir en un mayor costo y en muchas ocasiones, este costo no se toma en cuenta o no tiene la importancia necesaria. Cuando se inicia el proyecto, si no se obtienen los requerimientos de la manera más productiva, que sería usar todo el tiempo disponible de los involucrados para entender los requerimientos, es posible que se pierda tiempo en identificar el problema o inclusive se puede correr el riesgo de entregar un producto inservible por obtener malos requerimientos. Al mismo tiempo, es muy importante que los desarrolladores sean lo más productivos posible y que siempre estén ocupados, ya que cada momento que no estén realizando alguna tarea, es tiempo ocioso y dinero que se está perdiendo.

Al momento de comenzar el desarrollo de un software, también se tiene que elegir bien la arquitectura que se va a utilizar, elegir el recurso humano más adecuado, las mejores computadoras y demás componentes involucrados en el desarrollo, desde el inicio, para

tratar de obtener la máxima productividad. Pero elegir a las personas más adecuadas y a las computadoras más rápidas no es suficiente, también, en el transcurso del proyecto, se tiene que evaluar la productividad constantemente. A la hora de hacer el desarrollo, es necesario evaluar que el software no tenga errores, que sea utilizable y estable, además de que esté escrito de la mejor manera, para que si alguien que no pertenece al grupo de desarrollo original tuviera que hacer cambios o mejoras, lo pueda hacer de una forma fácil. Claro, esto es muchas veces difícil de medir, ya que normalmente no se quiere invertir tiempo en evaluar el desempeño porque puede durarse mucho y porque el costo puede ser muy alto. A pesar de que hay muchas formas de controlar el desarrollo, no son fáciles de implementar.

También está el problema de que según quien hace la evaluación de la productividad, va a buscar medir lo que más le interesa. El gerente del proyecto va a querer medir un punto según las métricas que él necesite reportar a la alta gerencia, por ejemplo, si se va a terminar a tiempo el proyecto o si se está gastando dentro del presupuesto. El desarrollador, por otro lado, va a querer medir otro punto muy diferente para ver la eficacia del desarrollo, entonces, por ejemplo, va a medir líneas de código escritas o los defectos encontrados en las pruebas. El cliente por su lado, va a querer medir lo que le está costando, sin tomar en cuenta si se está haciendo bien o no el desarrollo, por lo que no va a saber si los recursos por los que está pagando están siendo utilizados de la mejor manera y dando lo más que pueden dar. Por lo que surge la duda: ¿cómo hacer para unir todas las diferentes perspectivas en una sola forma de medir la productividad, que responda a la necesidad de todos los interesados?

En general, medir la productividad de un equipo de desarrollo es difícil de hacer. Son muchos factores los que se deben de tomar en cuenta para poder medirla. Hoy no se cuenta con un modelo que dé una forma fácil de medir la productividad, por lo que muchos optan por hacer el proyecto, terminarlo lo antes posible y si hay tiempo, ver qué tan productivo fué el trabajo. Hay momentos en que se hacen evaluaciones para mejorar, pero ocurren al final del proceso, por lo tanto, si algo se hizo de la manera equivocada, ya no hay forma de corregirlo. Sería mucho mejor contar con un modelo que ayude a medir la productividad desde el principio del proyecto y hasta la finalización del mismo, de esta manera se podrían realizar cambios en el camino para tratar de mantener la mayor productividad posible. Para cubrir esta necesidad se crea el modelo de "días ociosos", descrito en esta tesis.

Existen varios modelos de desarrollo, como el de cascada, prototipo, espiral e incremental. En el modelo de cascada se obtienen todos los requerimientos al inicio del proyecto, se hace todo el desarrollo y luego se entrega el producto. Este modelo ha demostrado no ser el más recomendable, ya que en muchas ocasiones, el producto no es lo que el cliente solicitó o lo que necesitaba en un principio. En el modelo de prototipo, se hace un pequeño desarrollo no muy funcional para que el cliente lo vea y apruebe y luego se va agregando funcionalidad poco a poco, hasta que el cliente obtiene lo que quiere, pero requiere mucho tiempo y una muy alta inversión. En los modelos espiral e incremental sucede lo mismo, el cliente va recibiendo poco a poco el producto incompleto y se dura mucho tiempo en terminar el producto, además de que el cliente sólo está presente al inicio y al final del proceso.

Agile (Spencer, 2013) es una ideología donde el desarrollo de software se realiza en iteraciones repetitivas y donde se involucra al cliente durante todo el ciclo de desarrollo.

Agile sigue los 12 principios descritos por sus creadores en el Agile Manifesto (Beck, Beedle, Martin, Grenning, & Highsmith, 2013) y que se centran en dar el mayor valor posible al cliente. El cliente puede ir viendo cómo va avanzando el software, para así realizar pruebas e identificar posibles errores o problemas lo antes posible. Agile cuenta con varios modelos, que han nacido basados en los 12 principios. Dentro de los modelos encontramos Scrum (Schwaber & Sutherland, 2013) y Kanban (Anderson, 2010). Ambos se enfocan en estar trabajando en el requerimiento que mayor valor agregue al cliente. Los requerimientos en Agile se nombran "historias" (Cohn, User Stories Applied: For Agile Software Development, 2004) y cada uno cuenta con una prueba de aceptación. Scrum utiliza iteraciones fijas en tiempo, donde cada una de las historias, se debe completar y entregar o no es tomado en cuenta para calcular la velocidad del equipo. Esto significa que el cliente tiene que realizar todas las pruebas y aprobarlas para que sea considerada completa. En el caso de Kanban, no es necesario que al final de cada iteración la historia esté completa para que sea tomada en cuenta, ya que la velocidad se calcula por historia, desde que se inició a trabajar en ella hasta que se completa; luego se saca un promedio entre todas las historias. En el caso de Scrum se le da más importancia a terminar el trabajo de cada requerimiento para seguir con el siguiente, aunque esto signifique detener parte de los recursos momentáneamente, para terminar todo. En el caso de Kanban se enfoca más en mantener todos los recursos constantemente en uso, por lo que, para ser lo más productivos posibles, se utiliza mayormente Kanban.

Uno de los grandes problemas de todos los modelos, incluyendo Scrum y Kanban, es que ninguno cuenta con un modelo para medir la productividad de una manera que indique si el recurso está trabajando todo el tiempo y está produciendo constantemente. Kanban ayuda a

tratar que el recurso esté ocupado en su totalidad, pero no mide la utilización del mismo. Los modelos existentes tienen muchas formas de medir la calidad del software que se está construyendo, pero, una buena calidad no indica si el recurso se utilizó al 100% de su capacidad. La empresa donde se desarrolló esta tesis utiliza el modelo Kanban en los proyectos que se realizan, por lo que el modelo que se construyó fue diseñado para Kanban y las pruebas que se hicieron fueron en proyectos que utilizaron este modelo de desarrollo.

2. Objetivos

2.1 Objetivo General

El objetivo general de esta tesis es definir un modelo que permita medir la productividad durante todo el ciclo de vida del desarrollo de software, cuando se utiliza el modelo de desarrollo Kanban que pertenece a la ideología Agile. Este modelo describe los pasos a seguir para la correcta medición de la utilización de cada uno de los recursos que trabajan desarrollando un software. En este caso, cuando se dice recursos, se refiere a las personas que participan en el proceso de desarrollo. La productividad se va a medir en cada una de las etapas del modelo para un mejor manejo de los datos y poder evaluar constantemente la utilización del recurso. Con este modelo se puede llevar un mejor control de la productividad y sirve para comparar proyectos realizados y analizar si se está haciendo buen uso de los recursos, o sea un 100% del tiempo, o si al contrario, se necesita mejorar.

2.2 Objetivos Específicos

I. Definir los actores involucrados en el desarrollo de software que se va a medir.

Como se mencionó anteriormente, durante el desarrollo de software se involucran muchas personas en diferentes momentos del proceso. El modelo desarrollado va a

definir la mejor forma de medir a cada una de las personas involucradas en todo el proceso. Como no es lo mismo el trabajo que realiza un desarrollador al de un analista, el modelo va a definir una forma de tomar medidas a cada uno por separado, pero que permita comparar a todos por igual al final.

II. Identificar, para cada una de las etapas del desarrollo, qué se va a medir y cuándo se debe realizar la medición.

Agile define cuáles etapas se deben tener en los diferentes proyectos, el modelo va a indicar en qué momento de las etapas se va a tomar cada una de las medidas propuestas. Esto permitió comparar los resultados de los diferentes proyectos que se realizaron con Kanban.

III. Definir las medidas que se van a tomar en cada una de las etapas del desarrollo del software. Cada etapa y cada actor se deben medir diferente, pero de una forma que se puedan comparar, entonces el modelo define estas medidas que darán una productividad general y servirá para comprar diferentes proyectos.

IV. Demostrar la validez, eficacia y aplicabilidad del modelo "días ociosos", desarrollando varias aplicaciones usando el modelo de desarrollo Kanban de Agile, tomando las medidas propuestas para calcular la productividad y evaluando su utilidad.

3. Justificación

La productividad indica si se está haciendo un buen uso de los recursos o si están siendo mal utilizados, por ejemplo poner a un desarrollador en tareas con tecnologías que no conoce o que no domina. Además puede revelar si el recurso es bueno o malo, osea, si el recurso hace bien su trabajo en un tiempo acorde con lo que se esta pagando por el. Un mal

uso de los recursos lleva a que se gaste tiempo y dinero haciendo algo que va a terminar mal o que no va a cubrir las necesidades reales del cliente. Por esto, se debe de tratar de corregir cuando se están realizando las tareas y no esperar hasta el final para evaluar si se pudo hacer mejor. Hoy en día existen muchas formas de medir la productividad de los desarrolladores de software, también existen formas de medir la productividad de los analistas, inclusive de los gerentes, pero no se tiene una forma de comparar estas mediciones entre ellas, ni se tiene un modelo que ayude a unir todo. Si se logra conseguir una forma de comparar las mediciones, se va a poder tener un marco para corregir el proceso antes de que salgan mal.

Hoy es bien sabido que el desarrollo de software se hace más importante cada día, ya que ayuda a mejorar la productividad de las empresas, mejorar procesos y ayuda en todos los ámbitos de los negocios. Si se logra hacer que se utilicen los recursos al máximo, todo el tiempo y constantemente, las empresas podrán terminar los proyectos más rápido, permitiendo hacer nuevos proyectos. Siempre se quiere terminar lo más rápido posible, pero sin que posteriormente se generen gastos por arreglar errores en el sistema, que se produjeron por correr al empezar el proyecto. Para esto se usa el modelo como guía, para hacer el desarrollo del software bien, desde el inicio del mismo. Existen muchas formas de medir si el software es de calidad, pero la gran pregunta es si se realizó bien, si se utilizaron todos los recursos de la mejor forma y todo el tiempo que estuvieron disponibles.

Siguiendo un modelo de mediciones, se va a poder medir la productividad desde el principio, desde que se empezó a trabajar y esto va a ahorrar mucho dinero al final del proyecto, porque se va a poder usar el recurso de la mejor manera posible. Utilizando un modelo de medición de productividad, se puede identificar que algo no marcha bien o que

alguien no está trabajando, porque se tienen los pasos que se deben de seguir y cómo se deben de hacer las tareas, para así poder cambiar en el momento que se identifica el problema, lo que al final, va a ahorrar tiempo y dinero. Si uno de los miembros del equipo no hace bien las tareas, es probable que va a hacer que los demás también se equivoquen y tengan problemas, generando un mayor gasto, ya sea corrigiendo errores posteriormente o teniendo que reescribir el código. Esto aplica para todos los miembros, entonces si se logra identificar cada uno de los puntos a medir usando un modelo, se va a poder llevar un buen control de los recursos.

4. Antecedentes

Con sólo poner la palabra productividad en un buscador de internet, se obtienen una amplia gama de formas que existen para medirla y un gran número de autores e investigadores que explican cómo implementar las mediciones. Cada una de las investigaciones explica una forma diferente de medir, pero ninguna explica una forma de unir todos los diferentes métodos, por lo que no hay una forma de calcularla desde el inicio del proyecto hasta el final, comparando todos los componentes involucrados en el desarrollo de software. A continuación, se hace una síntesis de las diferentes publicaciones importantes que existen y se explica porqué se considera que fallan a la hora de medir completamente la productividad.

El Dr. Kan (2002), encargado de la calidad de software de IBM, describe muy bien, en el capítulo 4 de su libro, las diferentes métricas que se pueden tener para medir la calidad del software, punto que va muy directamente relacionado con la productividad. Él describe que hay tres categorías de mediciones, las de producto, las de proceso y las de proyecto. Todas en conjunto sirven para medir la calidad en general de todo el equipo. Estos mismos

conceptos se pueden y deben aplicar a la medición de la productividad de un equipo de desarrollo de software. Se tiene que tomar en cuenta el producto que se está creando, así como las personas que lo están haciendo y el proceso que se está utilizando para construirlo. En cada una de estas categorías se tienen que tomar medidas de productividad para evaluar el desempeño y así poder ver cómo se encuentra el equipo de trabajo.

En su investigación, Sudhakar (2010) hace un repaso por lo que significa un equipo de trabajo y algunas de las etapas de desarrollo de software que existen hoy en día. Esto da la base para saber por dónde empezar a hacer una investigación de productividad, da los límites de a quiénes involucrar dentro de la medición y de cuándo empezar y terminar de medir. También empieza a indicar un poco los tipos de mediciones que se pueden hacer, de las que se tienen medidas objetivas y medidas subjetivas. Las medidas objetivas van a ayudar a medir la calidad del trabajo realizado como medidas de complejidad del código, variantes de tiempo y variantes de código (Na, Simpson, Li, Singh, & Kim, 2007). Las medidas subjetivas son las que dan una medición de productividad de las personas y sus procesos, como el crecimiento en conocimiento de las personas, efectividad como equipo y viabilidad del sistema (Bahli & Büyükkurt, 2005). Otros tipos de medidas subjetivas son efectividad del producto y efectividad del proceso (Wallace, Keil, & Rai, 2004). Sudhakar, Farooq, & Patnaik (2012) también explican estas y otras formas de medir la productividad, lo cual está muy bien, pero no indica cómo hacer para unir todas las medidas en una sola que indique realmente si se está haciendo un buen uso de los recursos involucrados como un todo, lo que impide tomar acciones correctivas a corto plazo para corregir problemas.

De la misma forma, Card (2006) en su investigación, describe algunas de las diferentes formas de tomar medidas de productividad y cómo estas pueden ayudar con la calidad.

Empieza con la definición que dan algunos de los estándares que tiene la industria, como la IEEE con su estándar 1045 (IEEE, 1993), medición de productividad del software, que da una medición utilizando líneas de código y puntos de función. También está ISO/IEC 15939 (El Emam & Card, 2002) que utiliza un modelo de procesos que indica qué se tiene que medir y un modelo de información para las diferentes medidas que se deben tomar para la correcta medición de la productividad. Pero como él dice, ninguno de estos estándares indica qué medidas se deben aplicar según el modelo de desarrollo que se va a utilizar. Continúa describiendo algunas de las métricas que se tienen hoy, como lo son las líneas de código, puntos de funciones y medición de tamaño, pero en cada una describe porqué no son buenas, por ejemplo la medición de tamaño no da una verdadera indicación de qué tan grande fue el esfuerzo de desarrollo o porqué si se hizo un software grande no necesariamente fue difícil de hacer. Por otro lado, las líneas de código sólo se pueden medir al final del proyecto, cuando todo ya está escrito, entonces no sirven para hacer correcciones durante el proceso. Termina concluyendo que existen muchas maneras de medir la productividad, lo cual es cierto, pero indica que no hay forma de unir las diferentes mediciones, lo cual no necesariamente es cierto, ya que con una buena investigación se puede encontrar una forma de medir la productividad en todas las áreas necesarias.

Kitchenham & Mendes (2004) hacen un análisis más profundo sobre la medición de la productividad basada en tamaño, utilizando una fórmula donde la productividad es el resultado de dividir el tamaño entre el esfuerzo, dejando de lado la fórmula de entradas entre salidas. El problema que tratan de resolver, es que el tamaño puede medirse de muchas formas, lo que hace que se tenga que utilizar una fórmula para cada ocasión, pero

proponen aplicar un tamaño ponderado entre las diferentes mediciones: proponen utilizar horas como una medida estándar entre todos los involucrados del proyecto, toman las horas de investigación y las suman a las de diseño, desarrollo, implementación y todas las otras involucradas en el proceso y utilizan esto para medir el tamaño. El gran problema que se presenta con este modelo, es que la unidad de medición "hora" no dice si se está haciendo un uso completo de los recursos o si están ociosos en algún momento del proceso, simplemente indica si se está durando mucho o poco, asumiendo que durar menos tiempo significa que es un buen desarrollo. Pero el hecho de durar menos tiempo puede significar que no se hicieron algunos pasos necesarios, como pruebas automatizadas, reutilización de código, y otras prácticas que se pueden implementar, por lo que no indica realmente si se hicieron de la mejor forma posible y usando todos los recursos disponibles al 100%. Una buena medida de productividad debería indicar si el código y los requerimientos son buenos y no solamente si se terminó rápido.

Buscando una visión diferente, Ryman & Reddy (2009) escriben un resumen de cómo la forma en que los altos ejecutivos hacen mediciones de sus equipos de trabajo, se puede llevar a los equipos de desarrollo de software. Ellos hablan de como el retorno de la inversión casi nunca se da porque en la gran mayoría de los casos el software no se termina a tiempo, hay retrasos y/o el cliente no queda satisfecho con el producto final. También mencionan cuáles son los principales problemas a la hora de medir la productividad, como grupos distribuidos, colaboración difícil entre organizaciones internas y estructuras heterogéneas dentro de las organizaciones que dificultan la unidad. Luego explican que muchas de las mediciones actuales son hechas a mano y que esto puede llevar a que se pierdan muchos detalles, como que una parte del código no esté probada, lo que puede

ocasionar defectos una vez implementado el software y que esto no se toma en cuenta para medir la productividad. También presentan una tabla comparativa donde muestran un incremento en casi un 50% de efectividad cuando se toman medidas de productividad desde el inicio del proyecto. Al final, lo que ellos proponen es que la recolección de los datos sea totalmente automática y que se deben de tener alertas que indiquen cuando las mediciones que se están tomando no van bien, lo cual genera dos problemas. El primero es que no todas las medidas se pueden tomar automáticamente, por ejemplo, si se quiere medir la calidad de los requerimientos, esto es muy subjetivo y no se puede dejar a una computadora, por lo menos hoy, que mida calidad. El otro problema es que no indican cuáles medidas se deben usar, sólo dicen que hay que medir algo, pero no indican qué, lo que resulta muy ambiguo y sujeto a la objetividad del lector, ya que es muy variable la cantidad de medidas que pueden existir y que se pueden tomar automáticamente.

Por otra parte, Numrich, Hochstein, & Basili (2005) realizan una investigación donde se define un espacio métrico para poder evaluar a los desarrolladores de un software. Este espacio permite hacer comparaciones entre ellos para encontrar cuáles son los que hacen mejor el trabajo y así cambiar al que no lo hace bien o emparejarlo con uno que lo hace bien, para que aprenda de este. Esto también va a permitir tener un mejor equipo de desarrollo, que en teoría podría escribir un mejor código, aunque esto no siempre es cierto, ya que si se comparan dos personas que son malas, se podrá saber sólo cuál es la menos mala, pero realmente no se está midiendo su verdadero desempeño. El otro problema que plantea esta investigación, es que sólo se enfoca en el desarrollador, cuando se sabe que el hacer software va mucho más allá de estos. Que si bien es cierto, son los que más trabajo harán en el proceso, no son los únicos involucrados. Por ejemplo, si los desarrolladores

tienen malos requerimientos pues el software será malo, entonces también es importante unir el desempeño de los desarrolladores con el de los demás miembros del equipo de trabajo.

5. Hipótesis

La ideología de desarrollo de software Agile está muy bien estructurada, define cada una de las etapas a seguir cuando se hace software y los actores que participan en cada una, pero realmente no define cómo medir la productividad en cada una de las etapas, ni para cada uno de los involucrados. Existen muchas formas de medir la calidad del software y la calidad del trabajo que se realizó, pero ninguna demuestra si se está utilizando el recurso de manera productiva o si estuvo ocioso en algún momento. Si se logra identificar una forma de medir los días que se perdieron por culpa de los problemas comunes durante el desarrollo, como cambios en los recursos o cambios en los requerimientos y que además tome en cuenta los días que los recursos estuvieron ociosos, se puede medir cuál fue la verdadera productividad del proyecto.

El modelo “días ociosos” propuesto en este documento, ayuda a medir la productividad de principio a fin. Este modelo indica, paso a paso, cuándo y cómo tomar medidas para calcular la productividad en cada una de las etapas y para cada uno de los actores involucrados en el desarrollo de software, usando Agile y Kanban. A su vez, indica cómo se van a comparar cada una de las medidas que se van a tomar durante el desarrollo del software, para sacar un solo valor, que revele si todo el equipo es productivo. Estas medidas, además, dan un marco de referencia para comparar diferentes proyectos y equipos de trabajo y así identificar dónde se puede mejorar, en comparación con otros proyectos o equipos que desarrollan software.

6. Metodología

La investigación realizada para esta tesis se divide en dos partes, la definición del modelo y las pruebas realizadas. Para la definición del modelo, primero se realizó la investigación descriptiva documental, o bibliográfica, de Agile y del modelo de desarrollo llamado Kanban. Se realizó un gran estudio de los aspectos generales de este modelo para entender a fondo los alcances del mismo, entender su funcionamiento y poder encontrar dónde se pueden presentar problemas en los modelos de medición. Además, se buscó una solución al problema de la productividad. Una vez que se tuvo la información, se procedió al desarrollo del modelo de medición de la productividad, utilizando un método empírico-analítico basándose en la experiencia de todas las personas involucradas en el desarrollo de software donde se realizó la investigación. Todos los involucrados tenían más de 5 años de experiencia en el campo, por lo que tenían un gran conocimiento de las cosas que afectan los proyectos de desarrollo de software. El fin de esta investigación fue proveer una solución al problema de baja productividad en proyectos de desarrollo que utilizan Kanban. De esta investigación sale el método de Días Ociosos.

Con el modelo de medición ya definido, se procede a realizar la segunda etapa, que es una investigación de campo utilizando un método de observación científica. Esta investigación contó con la realización de tres proyectos piloto de dos meses de duración, utilizando la ideología Agile y su modelo de desarrollo Kanban, incorporando el nuevo modelo de medición definido en esta tesis para evaluarlo y comprobar su validez y aplicabilidad. Durante este tiempo se fueron tomando los datos requeridos para la investigación que llevarían a las conclusiones de esta tesis.

Al ser el tiempo para terminar la tesis muy corto, solamente de ocho meses y contar con muy pocos recursos para realizar la investigación, fue difícil establecer la verdadera validez del modelo, principalmente por dos motivos. Primero, la población para una investigación como esta es muy grande, se puede decir que la población son todos los equipos de desarrollo de software que existen y la muestra que se está tomando se podría considerar insignificante, ya que fue sólo un equipo de desarrollo el que se midió. Segundo, por lo general, un modelo de medición dura mucho tiempo en establecerse, se puede ver con el caso de Agile, que inició hace ya más de 10 años y todavía hay gente que duda de su validez. Lo importante de destacar es que el propósito principal de la tesis fue llevar a cabo el objetivo general, que es definir el modelo y comprobar su aplicabilidad. Una vez que se empiece a utilizar en más proyectos, se irá concluyendo si los puntos planteados son correctos y si tienen una verdadera utilidad. Al final de cada proyecto se evalúa el modelo y se muestran los resultados de la productividad obtenida.

7. Cronograma

La tesis se dividió en tres etapas principales. La primera fue de cuatro meses, en los cuales se definió el modelo. Durante este tiempo se realizó la investigación, se desarrolló y escribió el modelo y se definieron las mediciones del mismo. La segunda etapa consistió en realizar los tres proyectos con Kanban, incorporando el modelo definido y al final de cada uno se realizó una reunión para medir la validez y aplicabilidad del modelo. Por último, la tercera etapa fue la documentación de los proyectos, resultados, conclusiones y recomendaciones.

Los cuatro meses de investigación y desarrollo se distribuyeron de la siguiente forma:

- 15 días para definir las etapas del desarrollo que se agregarían al modelo
- 15 días para definir los recursos que se iban a medir utilizando el modelo
- 1 mes para definir las mediciones que se querían tomar para cada etapa y para cada recurso del proyecto
- 15 días para definir en qué parte de cada etapa y cómo se iban tomar las medidas a utilizar
- 1 mes para definir cómo se unirían todas las medidas para obtener un valor final
- 15 días para finalizar la documentación y presentarla al equipo de trabajo que las iba a ejecutar

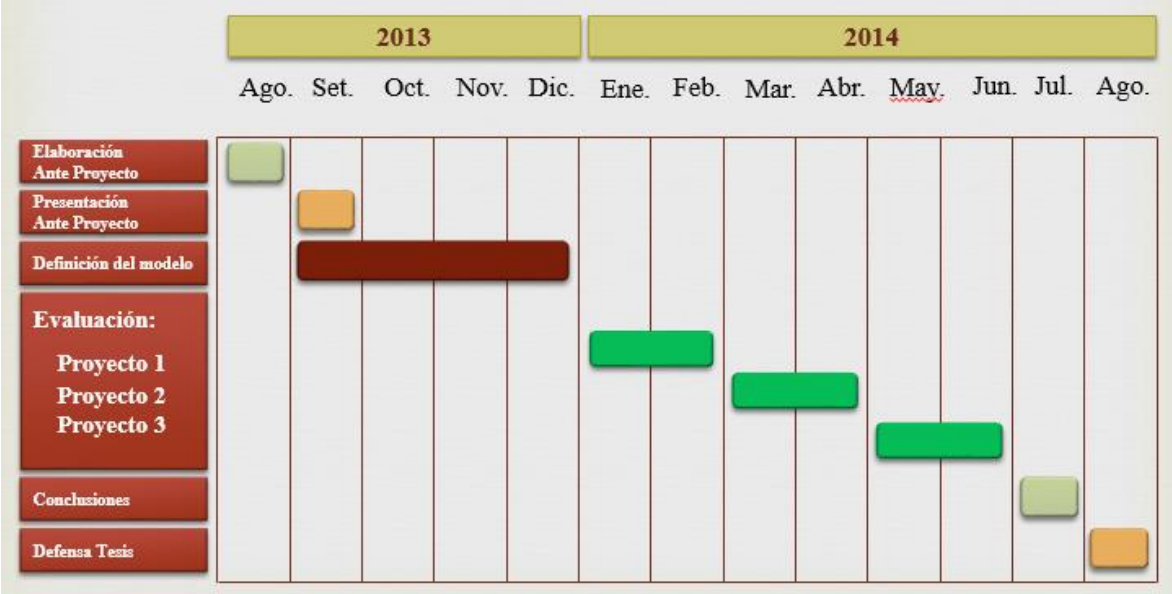

Figura 7.1 – Cronograma de la tesis

8. Recursos Humanos

El desarrollo del modelo fue realizado con la ayuda de la profesora tutora y varias personas que conforman la comunidad Agile de la compañía donde se realizó. Una vez concluida la fase de investigación y definición del modelo, se definió un grupo de desarrollo de

software, que fue el grupo que realizó tres proyectos pequeños para el departamento de Servicios Corporativos de la compañía.

El grupo de trabajo que se utilizó es un grupo ya establecido, con amplia experiencia en desarrollo de software, que ha estado trabajando junto por más de 5 años. Cuenta con un gerente de proyecto, que es el encargado de manejar los recursos, el tiempo y la resolución de conflictos del proyecto a desarrollar. Él es el interesado en que el modelo le suministre las medidas adecuadas de productividad. También está conformado por dos analistas de sistemas encargados de obtener los requerimientos y de velar porque el software que se está desarrollando cubra las necesidades del cliente. Ellos buscan la mayor productividad en las etapas iniciales del proyecto y a la hora de transferir los datos a los desarrolladores para su implementación. Además, se cuenta con 6 desarrolladores de software, que son los encargados de hacer el trabajo de codificación y qué tan productivos son.

9. Recursos Materiales

Al tratarse del desarrollo de un modelo, no se requirió en la primera etapa de ningún recurso material, ya que sólo fue necesaria una amplia investigación y tiempo para definir la misma y sus detalles. Para la segunda etapa, que fue la evaluación del modelo, se requirieron los recursos necesarios para ejecutar los tres proyectos de desarrollo de software, como lo fue el equipo de cómputo y el grupo de trabajo ya mencionado.

10. Evaluación económica

Al tratarse de un proyecto de interés para la compañía y ser el equipo de desarrollo que normalmente se usa en el desarrollo de software, no fue necesario invertir más dinero del

que ya se le estaba pagando a recursos, por lo que toda la parte económica fue cubierta sin problemas.

11. Definición del modelo

La ideología Agile ha sido muy bien aceptada en la industria, porque toma en cuenta la mayoría de las etapas necesarias y define claramente el proceso de desarrollo de software, además de que indica muy bien las personas que participan en el proceso y define los roles que cada persona debe de tener. Lo único con lo que Agile no cuenta es con un modelo que ayude a medir la productividad. Si bien es cierto, lo más importante de desarrollar un software es la satisfacción del cliente, también debería de interesar que se haga de la mejor forma posible, utilizando adecuadamente los recursos con los que el proyecto cuenta y evitando el desperdicio. En las próximas secciones, se describe cada uno de los actores involucrados, las etapas que se deben de tener al utilizar la ideología y se definen las diferentes formas de medir la productividad de todo el proceso. Luego, se describe cómo unir todo para poder hacer comparaciones con otros proyectos del mismo grupo e inclusive comparar diferentes grupos de trabajo.

11.1 Definición de los actores y medidas

Cuando se desarrolla un software hay muchas personas involucradas y cada una cumple un rol dentro del proceso. El cliente es el punto de partida, cuando indica lo que necesita. Es además el punto de salida cuando obtiene el software que satisfaga sus necesidades y le facilite o mejore su trabajo. Además del cliente, se cuenta con analistas de sistemas, un gerente de proyecto, un scrum master y uno o varios desarrolladores de software, dentro de los cuales está el líder técnico. Cada actor es responsable de cumplir con su parte para que

todo salga bien y así salir todos adelante como equipo. A continuación se va a describir cada uno de los actores, sus funciones y las posibles formas de medir su productividad.

11.1.1 Cliente o dueño del producto

El cliente es el actor principal, es el más importante, es quien paga para que se desarrolle el software y que espera que se le dé el producto que solicitó terminado, que cumpla todas sus expectativas, por eso se le dice también, el dueño del producto (Galen, 2009). El cliente será quien tiene un proceso que quiere mejorar y espera que el software le cubra la necesidad o le sirva para poder reducir los costos de su negocio. Sea cual sea el motivo del cliente, él es quien dará los requerimientos, o historias en Agile, y espera que el equipo de desarrollo le entregue el producto en el tiempo más corto, haciendo el mejor uso de los recursos y siendo lo más productivo posible. En la ideología Agile, el cliente es tan responsable del éxito del proyecto como lo son los demás miembros del grupo de desarrollo.

Cuando una persona construye una casa, va a ir todos los días al sitio de la obra para ver cómo marcha el proceso, que le cuenten cómo se encuentra la obra, y hacer inspecciones del trabajo realizado. Además, conforme van quedando listas las partes de la casa como el agua, luz y demás, también va a querer ir probando que todo funcione correctamente. En la ideología Agile es ideal que el cliente haga lo mismo, como se le está construyendo un software, se espera que venga todos los días a inspeccionar el proceso y ver que todo lo que pidió se esté haciendo según lo que él necesita, y que entienda cómo se va formando el software. La ideología indica que se tiene que tener una reunión diaria, en la mañana, donde los desarrolladores tienen un espacio para decir qué hicieron el día anterior, qué van a hacer durante el día para que el cliente esté enterado de cómo se están usando los

recursos y que vea que están haciendo algo en todo momento, ya que para eso está pagando. Durante esta reunión, también, se da un espacio para que los analistas o desarrolladores hagan consultas sobre dudas que puedan tener de alguno de los requerimientos. Otra expectativa que se tiene del cliente, es que esté disponible para consultas en el transcurso del día, en caso de que haya una pregunta que no pueda esperar al día siguiente. Pensando igual, los desarrolladores no deberían estar sin trabajar en ningún momento, ya que todo el tiempo que están ociosos es tiempo perdido.

No se quieren tener personas ociosas o esperando a que les asignen trabajo, entonces se necesita que el cliente tenga algo listo en todo momento. Si alguien del equipo tiene una duda y tiene que esperar al día siguiente para resolverla, entonces va a generar atrasos en el proceso. A este atraso se le va a llamar "día ocioso" y se va a agregar en una lista para su control. Inclusive, si el cliente un día no llega a la reunión y su presencia era requerida, porque hay problemas que necesitan su atención, se van a perder más días. Agile ayuda a contrarrestar esto teniendo más de una tarea lista para desarrollar en caso de que alguna de las tareas no se pueda completar, evitando así que estén ociosos, pero habrá momentos en los que esto no será posible y se tendrán días perdidos y esto es lo que se usará en el modelo para medir la productividad.

Continuando con las responsabilidades del cliente, también debe supervisar el progreso de la construcción para identificar posibles errores de diseño o de entendimiento por parte de los demás involucrados. En algunas de las reuniones se le harán demostraciones al cliente de los módulos que se han construido, para que él les dé su aprobación y que no haya sorpresas al final del proyecto. Si por alguna razón el cliente se da cuenta de que hay un problema con una de las historias y la quiere cambiar o eliminar por completo, entre más

temprano mejor, ya que cada día que se invirtió podría también considerarse tiempo ocioso, por lo que esto también se va a medir. Cada vez que el cliente decide cambiar algo en lo que ya se trabajó, se va a tomar como tiempo ocioso y se va a ir guardando para sumar a la medición de la productividad. Para esto se va a agregar al modelo una categoría que indique los diferentes tipos de días ociosos, por ejemplo, atrasos por falta del cliente será la primera y tener que realizar el trabajo de nuevo será la segunda de las categorías.

Una responsabilidad más del cliente, dentro de la ideología Agile, es hacer pruebas a los módulos que se van terminando. Cada vez que los desarrolladores terminan una de las historias, el cliente debe realizar las pruebas para asegurarse de que todo lo que pidió se cumpla y todo lo que necesita esté presente. La ventaja de la ideología es que al ser iterativa, el cliente se va a dar cuenta temprano de que algo no está como lo quería y se va a poder corregir a tiempo. Y así llegamos a nuestra tercera parte de la medición: cada vez que el cliente encuentra un error en la aplicación, se va a necesitar un tiempo para solucionarlo, en este caso, el tiempo que se va a tener que tomar en cuenta va a ser el tiempo necesario para solucionar el problema, ya que la mayor parte del trabajo ya se realizó y sólo hacen falta algunos cambios. También esta es la tercera categoría de errores. Por la naturaleza de la ideología y el involucramiento del cliente en todo el proceso, no debería ocurrir que toda la historia se tenga que eliminar completamente, pero si esto se da, sería probablemente un error en la obtención de los requerimientos y esto se va tomar en cuenta en la próxima sección.

Si bien es cierto que a pesar de que el cliente es quien está pagando y puede decidir cambiar de opinión o no asistir a las reuniones bajo su propio riesgo, lo que se pretende es medir la productividad del equipo de desarrollo, por lo que se tienen que tomar en cuenta

todos los atrasos, o días ociosos, para el modelo. También es importante recordarle al cliente la necesidad de su presencia constante en el desarrollo, para evitar al máximo tener demoras por culpa de malos requerimientos o errores que se hayan podido dar.

Resumiendo, del cliente se van a obtener las medidas de días ociosos en tres ocasiones, a la hora de hacer consultas sobre los requerimientos en la reunión diaria, cuando se hacen las demostraciones de los módulos en caso de que haya cambios y cuando se hacen las pruebas del trabajo realizado. Todo esto se va a medir en días que se llaman “días ociosos” y a cada uno se le va a agregar una categoría que indica qué tipo de día ocioso es, como atrasos, re-trabajo o errores. La medida de días ociosos es la que se va a utilizar como común denominador para todo el modelo.

11.1.2 Analista de sistemas

Los analistas de sistemas son los encargados de entender las necesidades del cliente y de escribir las historias que los desarrolladores van a completar. En Agile, cada historia va a tener lo que se llama una prueba de aceptación de cliente, que le indica a los desarrolladores, paso a paso, cada prueba que el cliente va a realizar para asegurarse de que se cumpla todo lo solicitado. Estas pruebas son las que los desarrolladores utilizan para hacer sus estimaciones, ya que tienen todas las especificaciones que se deben de realizar en el software. Es una ideología iterativa y durante una iteración se dan varias etapas al mismo tiempo; por lo que mientras los desarrolladores estén trabajando en unas historias, el analista está escribiendo nuevas historias y pruebas de aceptación para la próxima iteración. Ya que los analistas fueron los que obtuvieron los requerimientos del cliente, se supone que deberían de entender el negocio del cliente lo suficiente como para poder analizarlo y proveer mejoras en los procesos. Como el analista tiene este conocimiento, los

desarrolladores deberían de poder ir donde el analista para aclarar dudas, pero el analista puede que no tenga la respuesta y si no lo puede aclarar tiene que ir donde el cliente, nuevamente es posible que se pierda tiempo en esto y es importante anotarlo en la categoría de atrasos.

Al ser los analistas los que escriben las historias y las pruebas, si se comete algún error en la definición, es posible que los desarrolladores también lo hagan a la hora de hacer la programación, si esto ocurre y hay errores en las pruebas de aceptación también, se van a generar atrasos que se deben medir de la misma forma que los de los clientes, apuntando los días ociosos que se den.

Por otro lado, los analistas también se encargan de ayudarle al cliente, en ocasiones realizan algunas de las pruebas del sistema para ver que lo desarrollado cumpla con lo que se solicitó. Ambos deberían tener un conocimiento del negocio muy similar, entonces los analistas son la primera línea de defensa para que los errores no lleguen al cliente. La idea central de Agile es que al finalizar cada iteración, las historias deberían de estar probadas, si alguna de ellas no se probó, entonces también se van a generar atrasos para cerrar la iteración y esto también se debe de tomar en cuenta, ya que si se acumulan las historias para el final, se corre el riesgo de que los desarrolladores no tengan trabajo, pero no puedan decir que el proyecto se terminó porque no todo está probado. Por cada día que una historia esté lista pero no se pruebe, está atrasando las demás historias y este atraso se va a agregar a las mediciones del modelo. De igual forma, si a la hora de realizar las pruebas se encuentra algún fallo en la obtención de los requerimientos por parte del analista, se tiene que tomar en cuenta esto dentro del modelo, en especial si la historia se tiene que rehacer

del todo. Los días que se invirtieron en realizarla se agregarán a la lista de días ociosos del modelo como re-trabajo.

En resumen, las mediciones que se hacen a los analistas son iguales a las del cliente: en días ociosos por categorías, esto genera una forma de medir la productividad, igual para todos los que participaran dentro del desarrollo de software. Para los analistas serán las medidas sobre la validez de los requerimientos, la realización de las pruebas de aceptación y la facilidad de responder dudas de los requerimientos que los desarrolladores tengan.

11.1.3 Desarrolladores de software

El tercer grupo de actores para analizar son los desarrolladores. Son los encargados de hacer el software y darle forma a las historias y utilizan las pruebas de aceptación como su guía de trabajo para cada una. Para esto, lo primero que tienen que hacer es entender lo que el cliente solicita y hacer las preguntas necesarias para poder construir el software. Al mismo tiempo tienen que utilizar alguna técnica de estimación (Cohn, Agile Estimating and Planning, 2005) para ver cuán compleja es la tarea. La técnica más utilizada es aplicar unidades de esfuerzo, o puntos de esfuerzo usando una escala de fácil, mediana, difícil o muy difícil. A cada una de las categorías de la escala se le asigna un peso, por ejemplo las fáciles pueden ser 1 y las medianas 2. No se utiliza tiempo, sino que utilizan la complejidad, porque es muy difícil dar una estimación de tiempo realista, ya que hay muchas situaciones que la van a afectar, como las vacaciones, enfermedades, tiempo libre, feriados, otras tareas o responsabilidades, entre otros motivos. En cambio, la complejidad es igual en la mayoría de las situaciones, sin importar las circunstancias del equipo.

El concepto llamado velocidad, indica básicamente cuántos puntos se pueden realizar en cada iteración. Una vez que se decide una velocidad, por ejemplo, 8 puntos por iteración, se puede saber cuánto va a durar todo el proyecto, calculando cuántos puntos hacen falta y dividiéndolo entre la velocidad. La velocidad sirve también para saber cuánto trabajo los desarrolladores van a poder realizar en el tiempo que dure la iteración. Además, ayuda a los analistas a saber cuántas historias tienen que tener listas para la próxima iteración y le indica al cliente cuántas va a tener que probar en cada iteración. En el caso de Kanban, los puntos indican un aproximado de cuánto va a durar una historia y de igual forma, el analista toma esto para saber cuántas historias es necesario tener listas para que los desarrolladores estén siempre ocupados con trabajo. Por esas razones es que la estimación de los desarrolladores es tan importante, porque un problema con la estimación puede tener dos impactos negativos en el tiempo del proyecto. El primero es que una estimación más alta se traduciría en que se terminaría antes de tiempo, lo cual es bueno para el proyecto porque significa que se ganó un poco de tiempo, pero si el analista no tenía nada preparado para que los desarrolladores puedan comenzar una nueva historia, entonces podría ocasionar que no tengan trabajo y pasen ociosos algún tiempo. Esta es la primera de las mediciones que se tienen que anotar: los días que pasen ociosos los desarrolladores por no tener historias para trabajar, por motivo de una mala estimación. Así por el contrario, si los desarrolladores hacen una estimación muy baja, significa que van a durar más tiempo de lo previsto desarrollando la historia, por lo que al final de la iteración no van a estar listas todas las historias, lo que va a repercutir en el atraso completo del proyecto y el cliente por su parte, no va a poder realizar las pruebas cuando debe y todo se va a ver afectado, por lo

que se va a registrar esta mala estimación también como tiempo ocioso y se va a agregar a las medidas del modelo.

En un tema aparte, también se busca que el equipo de trabajo sea eficiente, por lo que se espera que los desarrolladores hagan bien su trabajo escribiendo buen código. Un software con buen código se puede describir como legible, reutilizable y que cumpla con los estándares previamente establecidos por el grupo de desarrollo. Para cumplir con esto, Agile cuenta con la programación extrema, que son 12 prácticas que ayudan a programar mejor. Una de estas es el desarrollo en parejas, así, hay dos personas escribiendo el código al mismo tiempo y hay depuración inmediata del mismo y se pueden detectar errores casi al instante y no se debe invertir mucho tiempo arreglando el código posteriormente. No siempre se puede realizar la programación en pareja, por lo que se sugiere también que existan reuniones de revisión de código, donde se toma una parte del código y se evalúa para que sea eficiente. Aquí es donde entra el actor especializado llamado líder técnico, que es un desarrollador elegido por ser mejor en las tecnologías que se están usando para desarrollar el software y así, los demás desarrolladores tienen alguien a quién acudir en caso de que no conozcan la tecnología. Si este líder técnico no es especialista en la tecnología y tiene que estar consultando o aprendiendo cada vez que le hacen una pregunta los demás desarrolladores, se van a generar días ociosos para los que estén esperando la respuesta, por lo que esto se va también a medir usando el modelo con la categoría de atrasos.

Como cuarta medida para mejorar el código, se tiene una técnica llamada refactorización (Fowler, Beck, Brant, Opdyke, & Roberts, 1999), que es básicamente seguir las buenas prácticas de desarrollo y mejorar el código, por ejemplo, eliminando métodos duplicados y

códigos que realicen las mismas funciones y centralizando el código en capas. Cada una de estas prácticas para mejorar el código tiene un costo y va, posiblemente, a requerir tiempo para reescribir el código, lo que, de nuevo, va a generar días de retraso, no necesariamente ociosos, porque realmente sí se está trabajando, pero son días que no se tenían en cuenta, por lo que formarán parte de las mediciones del modelo para clasificarlos como ociosos bajo la categoría de re-trabajo. Conforme se va mejorando en la escritura del código y haciéndolo más eficiente, el tiempo que se invierte en estas técnicas se va a reducir, ojalá hasta el punto que no será necesario realizar la refactorización.

Una ventaja más del trabajo en parejas, las reuniones de revisión de código y la refactorización, es que se va a ir generando una similitud de escritura de código entre todos los del equipo, hasta el punto en que todos van a escribir muy similar. Cuando los desarrolladores tienen distintas formas de realizar ciertas tareas o de escribir el código, es muy difícil para uno continuar con el código de otro, y por la naturaleza de lo que se está realizando, se van a dar momentos en los que se irán de vacaciones o salgan por algún motivo del proyecto, temporal o definitivamente, lo que puede generar retrasos. Cada vez que haya cambios en los recursos del proyecto y se tengan retrasos o se tenga que escribir de nuevo código, también se debe de anotar en la medición, para llevar la cuenta de cuántos días se perdieron en la transición, esperando que conforme el proyecto avanza y los desarrolladores van escribiendo código más similar unos con los otros, estos tiempos vayan mejorando.

En resumen, se puede decir que para los desarrolladores, el cliente y los analistas, los días ociosos son una medida congruente. Cada día que se pierde de desarrollo, ya sea por mal

diseño, mal desarrollo, falta de conocimiento o malas estimaciones, se debe incluir en el modelo de medición.

11.1.4 Gerente de proyecto

En muchas empresas se tiene el rol de gerente de proyecto, que normalmente se encarga de obtener los recursos para el proyecto, asegurarse de que sean bien utilizados todo el tiempo, de que todos los conflictos sean resueltos a tiempo y de reportar a los jefes, o alta gerencia, el estado del proyecto. También es el encargado de tomar decisiones de cambios de tiempo y procesos que puedan generar gastos. En Agile se cuenta con un Scrum master (Adkins, 2010), cuyas funcionalidades son muy similares a las de un gerente de proyectos. En la compañía donde se realizó la investigación para esta tesis, el gerente de proyectos es además el Scrum master, ya que no se cuenta con una persona dedicada solamente a ser Scrum master, por lo que los gerentes de proyecto cumplen ambas funciones. El Scrum master se encarga de velar por el cumplimiento de las prácticas Agile y se encarga de realizar las diferentes reuniones. Al modelo que se está planteando, se le agrega una nueva tarea: llevar el conteo de los días ociosos del proyecto, que sean utilizados para medir la productividad del equipo de trabajo.

Pero no sólo será el encargado de llevar la medición, sino que también de medir la productividad. Como se mencionó antes, el gerente de proyecto se encarga de resolver conflictos entre recursos, por lo que se puede medir su capacidad para resolver los conflictos usando una medida igual a la que se utilizó con los demás, la cual agrega los días ociosos que se den por motivo de falta de resolución de problemas como atrasos.

Como el gerente de proyecto debe manejar los recursos, se espera que de antemano esté enterado de cambios en el personal, ya sea porque una nueva persona va a ingresar al proyecto para ayudar o porque alguien va a salir temporal o permanentemente. En el caso de los nuevos ingresos, se espera que inicien su trabajo lo antes posible y es la expectativa de que el gerente de proyecto busque dónde colocarlo. Cada día que este nuevo recurso no sea utilizando adecuadamente, es un día más que se suma a la medida de días ociosos. Así mismo, cuando una persona sale del proyecto, el gerente tiene que buscar una persona nueva para suplantarla, por lo que se espera que el reemplazo ingrese antes de que salga la persona, pero si esto no se da, el tiempo que pase entre la salida del recurso y la entrada del nuevo, se agregará a la lista de días ociosos también como atrasos.

Se hace la aclaración de que el gerente de proyecto podría verse como juez y parte en esta situación, porque apunta los días ociosos y se le pueden poner días ociosos a él también, por lo que se va a confiar en la honestidad del gerente, pero además se va a realizar una revisión de los días ociosos en las reuniones de retrospectiva, para asegurar que todos los días ociosos sean realmente tomados en cuenta. Esto se explica con mayor detalle en la sección 11.2.5.

11.2 Definición de las etapas y medidas

Una vez definidas las personas que van a formar parte del equipo de trabajo, se dividen las tareas a realizar en etapas. La ideología Agile define las etapas que se deben de seguir cuando se desarrolla un software. La ideología además es iterativa, entonces cada vez que se termina una iteración, hay un proceso de retroalimentación donde se evalúa cómo le fue al equipo durante la iteración y se buscan mejoras. Pero en la definición de la ideología no se indica realmente cuándo medir la productividad. La reunión de retrospectiva es muy

objetiva y difícilmente va a medir la productividad, ya que los mismos participantes no van a querer comentar lo que se hizo mal o en lo que se falló. Cada iteración se compone de 5 etapas: análisis, diseño, desarrollo, pruebas e implementación. Cada una de estas tiene una funcionalidad, entradas y salidas, donde las salidas son las entradas de la etapa siguiente. Cuando se tienen las medidas que se pueden hacer para cada uno de los actores que participan en las etapas, a continuación se define en qué momento se toman cada una de las medidas.

Al inicio de cada iteración se lleva a cabo una reunión, llamada reunión “sprint”, donde primero se definen las historias a trabajar según las prioridades del cliente y que los desarrolladores van a programar según la velocidad anterior. Segundo, se van a definir cuáles serán las historias que se van a programar en la iteración posterior a la actual, así los analistas pueden ir preparando las pruebas de aceptación de las mismas. Una vez que se tiene todo lo anterior, se da inicio a la iteración y cada actor se va a hacer su trabajo.

11.2.1 Etapa de análisis

Esta es la primera de las etapas de la ideología, se da al inicio del proyecto y al inicio de cada una de las iteraciones. En esta etapa participan principalmente el cliente, trabajando en conjunto con el analista de sistemas, para analizar las historias que los desarrolladores van a trabajar en la siguiente iteración. En esta etapa también se definen las pruebas de aceptación para cada una de las historias y se documentan, para entregarlas a los desarrolladores. Es muy importante que el cliente esté muy involucrado para que el analista obtenga y document los requerimientos correctamente y a tiempo. El gerente de proyecto debe de estar muy atento a la forma en que están trabajando el analista y el cliente y resolver cualquier conflicto lo más pronto posible. Como ya se había mencionado, en esta

etapa es cuando se van a identificar los días ociosos que se puedan dar por culpa del cliente o del analista.

11.2.2 Etapa de diseño y desarrollo

Una vez que los desarrolladores tienen todas las historias que se van a trabajar, inicia la etapa de diseño donde, de manera simple, se esquematiza lo que se va a trabajar, para luego desarrollarlo. En esta etapa es donde los desarrolladores pueden cometer errores a la hora de hacer su trabajo, por lo que el gerente debe de estar muy atento y trabajar muy de cerca con el líder técnico, para identificar días ociosos y corregirlos lo antes posible, recordando que se pueden dar días ociosos por malos diseños, errores a la hora de hacer el código y hasta en las estimaciones.

11.2.3 Etapa de pruebas

Una vez que los desarrolladores completan la historia, se la remiten a los analistas para que corran las pruebas de aceptación previamente definidas. Es importante de notar que en esta etapa es cuando se puede dar un poco de ambigüedad, porque cuando el cliente dice que algo es un error o hace falta, es importante hacer un buen análisis para determinar si realmente es un error cometido por los desarrolladores o el analista, o si realmente fue que el cliente está solicitando algo que nunca se documentó. Esto, porque en muchas ocasiones los clientes dicen que hace falta algo de la historia, pero realmente fue que no lo solicitó desde un principio. En este caso no se debe tomar como un día ocioso, sólo se debe agregar una nueva historia a la lista de tareas por realizar. Por otro lado, si fue que el analista cometió un error y el cliente sí lo había solicitado o si los desarrolladores cometen un error, entonces sí se debe de tomar en cuenta para la medición.

11.2.4 Etapa de implementación

En cada una de las iteraciones se espera que se pueda hacer una implementación de lo realizado, claro, esto no siempre se da porque es posible que no se hayan realizado suficientes cambios que ameriten la implementación. Pero cuando sí sea necesaria, los analistas se encargarían de programar la implementación y coordinar con el cliente el mejor tiempo para realizarla. Una vez que se sabe cuándo se va a realizar la implementación, si lo amerita, el analista debería de notificarlo a los posibles usuarios afectados, para que sepan que el sistema estará fuera de línea por un tiempo.

11.2.5 Retrospectiva

Al final de cada iteración o al final de varias iteraciones juntas, según lo decida la carga de trabajo, se hace la retrospectiva (Derby & Larsen, 2013). En esta reunión se hacen tres preguntas principales: ¿qué se hizo bien y se debe seguir haciendo?, ¿qué se hizo mal y se debe dejar de hacer?, y ¿qué se debe comenzar a hacer para mejorar? Sumado a esto, se puede agregar la pregunta de cómo salió la medición, de cuántos días ociosos se dieron para llevar la cuenta por iteración. Así como se hace con la velocidad, que se mide y compara por iteración, se puede hacer con la cantidad de días ociosos. Esto va a ir ayudando a evitarlos en el futuro y si se ve que hay un área donde se está dando mucho el problema, se puede enfocar en mejorarla; ya sea con entrenamiento, con ayuda de otros o haciendo mejoras en el proceso. Lo importante es no dejar para el final el análisis, porque sino, ya no habría mucho para cambiar y mejorar.

11.3 Consolidación

El cuadro 11.2 contiene un resumen de las situaciones que se identificaron durante la investigación, no siendo las únicas existentes.

Actor	Categoría	Etapas	Situaciones
Cliente	Atrasos	Análisis Desarrollo Pruebas	No llegar a las reuniones donde es necesaria su presencia
	Re-trabajo	Desarrollo	Cambio en los requerimientos cuando se están desarrollando
	Errores	Análisis	Dar malos requerimientos o necesidades
Analista	Atrasos	Desarrollo Pruebas	No tener historias listas para los desarrolladores No conocer los requerimientos No conocer el negocio del cliente No realizar pruebas a tiempo
	Re-trabajo	Análisis Desarrollo	Documentación de las historias y pruebas de aceptación incompletas
	Errores	Análisis	Mala documentación de las historias y las pruebas de aceptación
Desarrollador	Atrasos		Malas estimaciones que lleven a no tener historias listas a tiempo Tener un mal líder técnico que no conozca las tecnologías que se están utilizando
	Re-trabajo	Desarrollo	Refactorización por tener un mal código
	Errores	Desarrollo Pruebas Implementación	Cometer errores al escribir el código No hacer una buena implementación de producto
Gerente de proyecto	Atrasos	Todas	No resolver bien los conflictos que se presentan Mal manejo de los recursos cuando ingresan o salen del proyecto

Cuadro 11.1 Posibles días ociosos de un proyecto

Se pueden dar otras situaciones que afecten la productividad, por lo que la lista que se presenta, es una guía de qué buscar en el proyecto, pero se tiene que estar pendiente de otras afectaciones que generen días ociosos.

11.4 Factor de afectación

Cada día ocioso que se obtiene involucra una cantidad diferente de personas, según el actor que lo ocasionó y según la situación que se dio. Tomando como ejemplo una historia mal documentada que hay que rehacer, esto va a afectar al desarrollador que realizó el trabajo original y al analista, pero no a los demás desarrolladores, que pueden seguir con su trabajo con normalidad. Por este motivo es que no se pueden contar todos los días iguales y se

tiene que aplicar un factor de afectación a cada situación, para poder contabilizar los días de una forma mucho más realista. Para esto se va a aplicar la siguiente fórmula de afectación:

$$\text{Factor de afectación} = \frac{\text{Total personas afectadas por la situación}}{\text{Total de personas que pudieron ser afectadas por la situación}}$$

Para cada uno de los días que se tienen ociosos, se divide la cantidad de personas involucradas o afectadas, entre la cantidad de personas que pudieron ser afectadas por el problema. El resultado de esta fórmula se multiplica por la cantidad de días que se afectó el proyecto y esto dará la cantidad real de días ociosos. Es importante aclarar que no se debe utilizar la cantidad total de personas del proyecto para calcular el factor de afectación, sólo a las personas que hubieran podido ser impactadas, esto significa que sólo se toman en cuenta las personas que están realizando algún trabajo en el proyecto en el momento que se dio la situación.

Por ejemplo, si el problema se dio a la hora de realizar una de las historias, el gerente de proyecto no se va a ver afectado ni se podría atrasar por la situación, igual que el cliente, pero sí se pudo haber afectado a cualquiera de los desarrolladores, por lo que sólo se tomaría en cuenta la cantidad de desarrolladores para hacer el cálculo. Por este motivo es que es importante hacer una buena evaluación de quiénes se pudieron ver afectados cuando se da un día ocioso, para no castigar de más al equipo y que sea lo más justo posible.

11.5 Tabla de control

Para facilitar la documentación y llevar un mejor control de los días ociosos, se debe utilizar una tabla de control donde se anota cada una de las situaciones que se dan, cuántos

días se atrasó el o los actores, la cantidad de personas que se pudieron ver afectadas y las que se afectaron, donde se pueda calcular el factor de afectación y sacar el valor real de los días ociosos. Se debe también anotar la fecha, el actor y la categoría de día ocioso que se presentó. Con esta tabla será mucho más fácil ir viendo por periodos de tiempo o iteraciones, cómo se está viendo impactada la productividad y tratar de reparar los errores lo antes posible. Al llevar este control e incluir la categoría, también se podrán sacar estadísticas por tipos de días ociosos, que ayudarán a encontrar los puntos más críticos que se deben de cuidar y solucionar lo antes posible.

Fecha	Actor	Etapa	Cat.	Días de atraso	Afectados	Posibles afectados	Factor de afectación	Días ociosos
01-Ene	Cliente	Análisis	Atraso	1	2	5	0.4	0.4

Cuadro 11.2 Tabla resumen de afectación

El cuadro 11.2 muestra la tabla de control y las columnas que la conforman. A manera de ejemplo, la tabla tiene los datos de la siguiente situación hipotética: el cliente no llegó a una de las reuniones y dos de los 4 desarrolladores se vieron afectados, ya que había preguntas que tuvieron que esperar hasta el día siguiente. El analista pudo haber sido afectado, pero en este caso no tenía ninguna pregunta, por lo que no se afectó, llevando el total de posibles afectados a 5. Al dividir 2 entre 5, se obtiene un factor de afectación de 0.4 que al multiplicarlo por 1 día ocioso, da un total de 0.4 días ociosos.

Cuando se quiere hacer una evaluación de cómo va la productividad, sólo se tiene que sumar la columna de días ociosos para obtener el valor que se va a utilizar para calcular la verdadera productividad, lo que dará la cantidad correcta de días ociosos que se obtuvieron en el periodo que se está evaluando.

11.6 Finalización del modelo

Ya se describieron las diferentes formas de medir los días ociosos, las categorías para dividirlos, cuándo tomar las medidas y cómo hacerlo a cada una de las personas que participan del proceso. En este modelo lo que se hace es comparar los días que se quieren evaluar, contra los días ociosos, dando como resultado la productividad. Se puede decir que un 100% de productividad sería que todos los días que se evaluaron, los recursos estuvieron ocupados todo el tiempo; lo que se busca obtener en todos los proyectos. Si a 100% se le resta el porcentaje equivalente a los días ociosos, o sea, que se perdieron por errores o problemas, se obtiene la verdadera cantidad de días que se utilizaron en el proyecto. A este porcentaje se le llama "porcentaje ocioso".

$$\text{Porcentaje Ocioso} = 100 \left(\frac{\text{Total días ociosos}}{\text{Total días proyecto}} \right)$$

Para calcular el porcentaje ocioso se toma la sumatoria de los días ociosos ya con su factor de afectación aplicado y se divide entre la cantidad de días que se han invertido. Luego se multiplica por 100 y esto dará el porcentaje de productividad negativa. Finalmente, se le resta al 100% para obtener la verdadera productividad. Esta sería la fórmula final:

$$\text{Productividad} = 100\% - \text{Porcentaje Ocioso}$$

Una variación de la fórmula sería hacer todo en un solo cálculo. Al total de días que se quieren evaluar del proyecto, se les resta el total de días ociosos, luego se dividen entre el mismo total de días que se quieren evaluar y finalmente se multiplica por 100. Así, se obtiene la productividad. Esta variación es la que se va a utilizar para realizar las pruebas.

$$\text{Productividad} = 100 \left(\frac{\text{Total días proyecto} - \text{Total días ociosos}}{\text{Total días proyecto}} \right)$$

Para demostrar las fórmulas y la aplicabilidad del modelo, se utilizará un ejemplo para un grupo de desarrollo de 7 personas: el cliente, el analista, el gerente de proyecto y cuatro desarrolladores. Se tiene una iteración de diez días. Se da una situación en una de las reuniones diarias donde el cliente no pudo asistir y hay dos desarrolladores que tienen una consulta importante y sin la respuesta, no pueden continuar con su trabajo. En esta situación, se agrega un día a la lista de días ociosos. Luego se le aplica el factor de afectación. En este caso, el problema con el cliente afecta a dos de las personas del grupo, pero los demás desarrolladores y el analista pueden seguir trabajando en lo que tenían en su lista de tareas y el gerente de proyecto no se toma en cuenta, porque se asume que no está tratando de resolver ningún problema en el momento. Con todo esto se puede decir que de las cinco personas que pudieron ser afectadas, dos realmente lo fueron, por lo que al aplicar la fórmula de afectación daría 0.4. Este resultado se le aplica al día ocioso que se perdió, multiplicando 1 por 0.4, que da como resultado que se perdieron realmente sólo 0.4 días. Como se tenía un total de 10 días y el cálculo da 0.4 días ociosos, se calcula la productividad restándole a 10 los 0.4 y luego se divide entre 10, lo cual da el resultado de 96%. Entonces, se puede ver que el atraso del cliente hizo que la productividad bajara un 4% durante los 10 días de la iteración. 96% es una muy buena productividad, pero si se permite que el cliente siga faltando, eventualmente el número va a ir creciendo, por lo que se tiene que hacer lo mejor posible por controlar todos los problemas que se puedan presentar, para evitar que se baje la productividad.

11.7 Escala de medición

No existe una referencia que indique qué es una buena o mala productividad para el modelo que se realizó, ya que ningún otro modelo de los investigados utiliza porcentaje de productividad. Por este motivo, también se propone utilizar la tabla 11.3 para indicar si una productividad es buena o mala, basada en la escala de muy mal, mala, regular, buena, muy buena y excelente.

Escala	**Productividad**
Muy mala	Menor a 50%
Mala	Mayor a 50% Menor a 70%
Regular	Mayor a 70% Menor a 85%
Buena	Mayor a 85% Menor a 95%
Muy buena	Mayor a 95% Menor a 100%
Excelente	Igual a 100%

Cuadro 11.3 Escala de productividad

- Igual a 100% o excelente es el mejor de los casos, es lo que se espera obtener en todos los casos, ya que significa no tener ningún desperdicio.
- Menor al 100% pero por encima de 95% se puede decir que es una productividad muy buena, por la alta utilización de los recursos.
- Menor a 95% pero mayor a 85% se puede catalogar como buena, porque se presenta una cantidad mayor de días ociosos que afectaron al proyecto, pero es manejable.

- Menor de 85% y mayor a 70% implica muchos días ociosos y el costo de estos está afectando mucho el valor del proyecto final, por lo que se puede concluir que fue un proyecto regular.
- Menor que 70% pero todavía superior a 50% se cataloga como malo, ya que indica que casi se perdió la mitad del tiempo del proyecto, por lo que ya se puede hablar de grandes problemas y es sumamente necesario encontrar una solución.
- Menor al 50%, o muy mala no debería de ser aceptada y se espera que nunca se dé; ya que es un indicador de que se desperdició más tiempo del que se trabajó y esto no debería ser aceptado, por lo que si la métrica va indicando una productividad tan baja, es mejor detener el proyecto y buscar cómo incrementar la productividad.

12. Pruebas realizadas

Para comprobar la aplicabilidad del modelo y evaluar su funcionalidad y veracidad, se implementó en tres proyectos a lo largo de veinticuatro semanas. Los tres proyectos de desarrollo de software se hicieron usando un grupo de desarrollo de software donde la ideología Agile se viene utilizando por muchos años y está bien establecida. Cada proyecto tuvo una duración de 8 semanas y cada uno tuvo variables interesantes que ayudaron a mejorar el modelo y comprobar su aplicabilidad. El equipo de trabajo cambió durante el periodo de tiempo, lo que trajo aspectos interesantes a considerar. A continuación se detallan los resultados de cada uno de los proyectos, se explica cómo se fueron dando los resultados y se presentan las tablas que se utilizaron, además se muestra el cálculo de la productividad para demostrar la aplicabilidad del modelo.

12.1 Primer proyecto

El primero de los proyectos tenía como propósito la implementación de mejoras a un sistema ya existente, de reservaciones de espacios en aviones de la compañía. El proyecto contó con la participación de un gerente de proyectos, una analista de sistemas y tres desarrolladores de software, además del cliente. El cliente sabía, a nivel global, cuáles cambios se querían, pero faltaban detalles por definir, por lo que la analista de sistemas se empezó a reunir con el cliente unos días antes del inicio del proyecto, para ir viendo los requerimientos y tratar de ayudar a que el cliente estuviera listo para el inicio del proyecto. El cliente tampoco había trabajado con Agile previamente, por lo que se le dio una capacitación previa para que se familiarizara con los conceptos y procesos. El proyecto inició el 6 de enero del 2014 sin problemas y todos los actores tenían trabajo que hacer. Se estimó que se duraría 8 semanas y finalizaría el 28 de febrero.

En la tercera semana se dio el cambio de la analista de sistemas, lo cual afectó mucho el proyecto, ya que la analista es quien tenía el conocimiento de la mayoría de las historias. El gerente de proyecto ya tenía el reemplazo listo, lo cual hacía suponer que no se iban a tener días ociosos, pero la analista de sistemas original no pudo darle un entrenamiento al nuevo, por lo que tardó dos días en aprender las historias, conocer el negocio y poder seguir con el trabajo donde la analista anterior había concluido. Se asumió que esto haría que se agregaran dos días ociosos, con un factor de afectación de 0.25, ya que los tres desarrolladores pudieron seguir trabajando sin problemas, pero cuando terminó la iteración, el analista de sistemas no tenía listas las siguientes historias para que los desarrolladores pudieran seguir trabajando, por lo que esto atrasó también a los desarrolladores. Los desarrolladores estuvieron tres días sin trabajo, mientras el analista

terminaba las siguientes historias. Entonces se agregó una entrada más a la lista de días ociosos que se tenía. Con esto se descubrió que cada vez que se da un caso donde se agregan días ociosos, es muy posible que se tengan que agregar más días para un actor diferente, por lo que cada vez que se da una situación, se tiene que seguir para que no se den más repercusiones.

Dado que se dieron estos atrasos, se tomó la decisión de traer a un desarrollador más para tratar de salir a tiempo con el proyecto y compensar los días perdidos. Como el proyecto ya había iniciado, todos los desarrolladores tenían sus computadoras listas para trabajar, pero el nuevo desarrollador no la tenía, por lo que el líder técnico de los desarrolladores tuvo que pasar un día completo explicándole los detalles del proyecto y configurando el ambiente de desarrollo. Esto hizo que se agregara un nuevo día a la lista de días ociosos, a pesar de que realmente no fue un día en el que no se trabajó, fue un día en el que uno de los desarrolladores no pudo avanzar en sus tareas. En este caso, como el nuevo desarrollador inició rápidamente a trabajar, le ayudó al que se había atrasado, por lo que no hubo mayor afectación.

Lamentablemente, el desarrollador tuvo que salir de la compañía dos semanas después, por lo que no pudo continuar. El gerente de proyectos indicó que no era posible reemplazar al desarrollador y que el proyecto debía terminar con tres desarrolladores. Esta situación hizo que se evaluara si se tenía que agregar algún tiempo a los días ociosos. Pero se llegó a la conclusión de que no era necesario, porque realmente no se iba a perder tiempo, simplemente se tenían menos recursos, por lo que no se podía hacer tanto trabajo, pero la productividad seguía siendo la misma, ya que ningún recurso quedó ocioso.

Continuando con el proyecto, la última iteración fue dedicada a los reportes del sistema. La alta gerencia quería que a todas las personas que habían tomado vuelos anteriormente o que tuvieran reservaciones a futuro, les llegara un correo electrónico con estos datos, además de su información de cancelaciones y costos que estuvieran relacionados con las cancelaciones. El problema fue que el cliente no supo explicarle al analista todo lo que quería en el estado de cuenta, sólo le indicó que quería mantener a los pasajeros informados. Por esto, los desarrolladores tuvieron que hacer un prototipo de reporte con la información que solicitó el cliente. La primera versión no fue del agrado del cliente y se tuvo que agregar tres días a la tabla de días ociosos, ante la necesidad de que el analista y los desarrolladores realizaran el trabajo nuevamente. Se le hizo una segunda versión al cliente que le gustó, pero a la hora de realizar las pruebas, el cliente solicitó que se le enviaran ejemplos con todos los escenarios posibles para validar los reportes, en varias ocasiones. Esto hizo que dos desarrolladores tuvieran que dedicar dos días a generar las pruebas, como estas pruebas no estaban en las pruebas de aceptación originales, se tuvo que involucrar al analista también para que documentara las pruebas. Estos días también se agregaron a la tabla de días ociosos, ya que fue algo que no se documentó bien a la hora de obtener los requerimientos.

Fecha	**Actor**	**Etapa**	**Cat.**	**Días de atraso**	**Afectados**	**Posibles afectados**	**Factor de afectación**	**Días ociosos**
20-Ene	Gerente de proyectos	Análisis	Atraso	2	1	4	0.25	0.5
27-Ene	Des.	Desarrollo	Atraso	3	3	4	0.75	2.25
03-Feb	Des.	Desarrollo	Atraso	1	2	5	0.4	0.4
19-Feb	Cliente	Pruebas	Re-trabajo	3	4	4	1	3
25-Feb	Analista	Pruebas	Atraso	2	3	4	0.75	1.5
							Total días	**7.65**

Cuadro 12.1 Días ociosos del primer proyecto de validación

El cuadro 12.1 muestra el resultado que se obtuvo de los días ociosos que se dieron durante el proyecto. Luego de aplicar la fórmula de afectación a cada uno de los eventos, se obtiene un total de 7.65 días ociosos. La fecha real de finalización del proyecto fue el 11 de marzo, lo cual muestra un atraso real de 7 días. Esto deja ver que el cálculo de días ociosos es muy cercano al atraso real, por lo que vamos viendo la validez del modelo.

$$\text{Productividad} = 100 \left(\frac{40 - 7.65}{40} \right) = 80.87\%$$

Cuando se aplica la fórmula de productividad se obtiene una productividad de 80.87%. Esto indica, según la escala establecida en la sección 11.7, que fue un proyecto regular, ya que se dieron muchos atrasos inesperados. Si se ve en tiempo, significa que el 20% fue ocioso. Tiempo que se hubiera podido invertir para entregar más trabajo o terminar el proyecto antes, para continuar con otros proyectos.

12.2 Segundo proyecto

El segundo proyecto que se utilizó para realizar las pruebas, fue hacer mejoras a un sistema ya existente, que se encarga de manejar los edificios de la compañía, toda la infraestructura, desde los cubículos de los empleados hasta las áreas comunes, como pasillos, escaleras y baños. Para este proyecto se contrató un trabajador nuevo, ya que no se contaba con todo el personal necesario para el proyecto. Este contó con un desarrollador de la compañía que le daba el asesoramiento. El proyecto tenía además un analista para trabajar con el cliente. En este caso no hubo un gerente de proyectos, por tratarse de un proyecto muy pequeño y subcontratado. Al ser un empleado subcontratado, se dio inicio al proyecto dándole una inducción del sistema actual y de la ideología Agile, así como los procesos que se utilizan en la compañía para desarrollar software. El proyecto inició el 3 de

marzo y se trabajó en la instalación de todas las herramientas de desarrollo. Para este proyecto se estimó como fecha de finalización, el 25 de abril.

En este primer día se dio el primer problema, ya que las licencias para las herramientas de software que cubren a la compañía, no cubrían a los trabajadores de contingencia, por lo que se tuvo que empezar por negociar la licencia con el dueño del software, para que no hubiera ningún problema legal. Los dos días que tomó esto, se tuvieron que agregar a los ociosos. Esto afectó también al desarrollador que estaba ayudando con la inducción, por lo que el factor de afectación fue de 0.67.

Una vez que el desarrollador tuvo todo instalado, se inició con las mejoras. En el primer mes de trabajo, el nuevo desarrollador necesitó la ayuda del desarrollador de la compañía en 3 ocasiones, para entender diferentes módulos del sistema original, pero no estaba disponible, por lo que en tres ocasiones se atrasó el avance del proyecto. Esto generó tres entradas en la tabla de días ociosos.

Una semana antes de que el proyecto se terminara se dio una situación particular. El cliente pidió dos cambios más que no estaban en la estimación original y que requerían la aprobación de la gerencia, ya que requerían que el tiempo del trabajador de contingencia se extendiera. Ya que el proyecto no contaba con un gerente de proyecto, se tuvo que empezar por conseguir todos los permisos para lograr la extensión, lo cual duró 3 días, pero el problema fue que esos tres días no se pudo trabajar en los requerimientos y hubo que agregarlos a la tabla de días ociosos. El cuadro 12.2 muestra el resultado final de todos los días ociosos que se presentaron.

Fecha	Actor	Etapa	Cat.	Días de atraso	Afectados	Posibles afectados	Factor de afectación	Días ociosos
03-Mar	Des.	Desarrollo	Atraso	2	2	3	0.67	1.33
13-Mar	Des.	Desarrollo	Atraso	0.5	1	2	0.50	0.25
18-Mar	Des.	Desarrollo	Atraso	1	1	2	0.50	0.50
24-Mar	Des.	Desarrollo	Atraso	0.75	1	2	0.50	0.38
28-Abr	Gerente de proyecto	Diseño	Atraso	3	2	2	1.00	3.00
							Total Días	**5.46**

Cuadro 12.2 Días ociosos del segundo proyecto de validación

En este proyecto se dio la situación antes mencionada de que se pidió más requerimientos al final, lo que hizo que el proyecto se hiciera más largo y los recursos fueran utilizados por dos semanas más de lo previsto. La duda era si a la hora de calcular la fórmula de productividad se debía tomar en cuenta la estimación original o si se tomaban en cuenta también los días extra que tuvo el proyecto. Se llegó a la conclusión de que era importante usar la duración completa del proyecto, porque era el tiempo que el recurso estuvo en utilización y de donde se iba a calcular el costo real del proyecto, por lo que sí era importante tomarlo en cuenta. El proyecto concluyó realmente el día 6 de mayo, lo que nos da un atraso real de 6 días, por lo que podemos ver que de nuevo, la cantidad de días ociosos de la fórmula es muy cercano a la realidad y podemos ir concluyendo que es válida.

$$\text{Productividad} = 100 \left(\frac{50 - 5.46}{50} \right) = 89.08\%$$

La productividad para este proyecto fue de 89.08%, lo que indica que fue un proyecto bueno siguiendo la escala de la sección 11.7. Esta productividad fue más alta y mucho mejor que la del primer proyecto. Igual no es 100%, que es lo que se espera, pero se mejoró la productividad y no se perdió mucho tiempo en solución de problemas.

12.3 Proyecto final

Durante el mes de abril se dieron grandes cambios en la compañía, por lo que se canceló el tercer proyecto que se iba a utilizar para la realización de pruebas y no había proyectos nuevos para realizar nuevas pruebas. Lo que se hizo fue agregar al equipo de trabajo a un proyecto que ya estaba en marcha y que tenía ocho meses de haber iniciado. Se empezó a tomar medidas en este proyecto el 5 de mayo del 2014 y se estimó concluiría el 27 de junio. Con la adición de varios desarrolladores, el proyecto quedó con nueve desarrolladores, un analista y un gerente de proyectos, además del cliente.

En las primeras semanas del proyecto se presentaron cuatro ocasiones en las que se dieron días ociosos, porque la tecnología que se utilizaba en el proyecto era muy compleja y era necesario que los desarrolladores más experimentados, ayudaran a los recién llegados a aprender la tecnología. Cuatro de las historias eran muy complejas y se requirió de la ayuda de los desarrolladores más experimentados para poner a todos al mismo nivel, lo cual atrasó a los desarrolladores que ya estaban en el proyecto.

En la quinta semana, se dio una situación con uno de los reportes. El cliente había solicitado un reporte muy especializado y se hizo utilizando una técnica que era necesaria para unir diferentes bases de datos al mismo tiempo, pero a la hora de ponerlo en producción no dio el rendimiento necesario y se tornó muy lento, por lo que hubo que rediseñar el reporte y volverlo a escribir, causando un atraso de cuatro días para dos de los desarrolladores que estaban trabajando en el reporte.

Por último, el día que se iba a poner parte de los módulos en producción, se presentó una situación con uno de los servidores donde se iba a instalar todo, por lo que el proyecto

sufrió un nuevo atraso de un día completo. Este atraso afectó a todas las personas involucradas, inclusive al cliente, porque no pudo realizar las pruebas necesarias ni dar inicio con la utilización del nuevo módulo.

Fecha	Actor	Etapa	Cat.	Días de atraso	Afectados	Posibles afectados	Factor de afectación	Días ociosos
05-May	Des.	Desarrollo	Atraso	2	6	10	0.6	1.2
08-May	Des	Desarrollo	Atraso	1	2	10	0.2	0.2
13-May	Des	Desarrollo	Atraso	1	4	10	0.4	0.4
14-May	Des	Desarrollo	Atraso	1	2	10	0.2	0.2
3-Jun	Des	Diseño	Re-trabajo	4	2	10	0.2	0.8
23-Jun	Des	Imple.	Errores	1	11	11	1	1
							Total Días	**3.8**

Cuadro 12.3 Días ociosos del tercer proyecto de validación

En este proyecto se da una situación a considerar. Como se puede ver en el cuadro 12.3, a pesar de tener más eventos dentro de la tabla, la afectación fue mucho menor, ya que gran parte del equipo no fue afectado, por lo que se minimiza el impacto de cada situación. Sólo se dieron 3.8 días según el modelo. La fecha real de finalización fue el 3 de julio, lo cual indica un atraso real de 4 días. Una vez más queda demostrado que el cálculo de los días ociosos es válido.

$$\text{Productividad} = 100 \left(\frac{40 - 3.8}{40} \right) = 90.50\%$$

La productividad calculada es de 90.50%. Una productividad buena, basados en la escala de la sección 11.7. Se puede ver cómo a la hora de agregar nuevas personas al proyecto, se va a tener una curva de aprendizaje que va a generar días ociosos, pero que conforme va avanzando el proyecto, esta curva disminuye y se van a tener menos problemas.

12.4 Otros Resultados

Los cuadros 12.4, 12.5 y 12.6 resumen para cada iteración de cada proyecto, cómo iba variando la productividad. Se puede decir que la productividad es muy variable en corto tiempo y que no es algo fácil de controlar, porque son situaciones que se presentan de improviso, pero que es muy importante que cada vez que la productividad baja, se utilice la reunión de retrospectiva para evaluar la situación y hacer lo posible porque no vuelva a suceder. También se puede anotar que no hay una relación directa entre las iteraciones y que las situaciones que afectan la productividad se pueden dar en cualquier momento del proyecto y lo mejor que se puede hacer, es tratar de minimizar el impacto.

Iteración	Días Ociosos	Productividad
1	0	100
2	2.75	72.5
3	0.4	96
4	4.5	55

Cuadro 12.4 Días ociosos por iteración, primer proyecto

Iteración	Días Ociosos	Productividad
1	1.58	84.2
2	0.88	91.2
3	0	100
4	3	70

Cuadro 12.5 Días ociosos por iteración, segundo proyecto

Iteración	Días Ociosos	Productividad
1	2	80
2	0	100
3	0.8	92
4	1	90

Cuadro 12.6 Días ociosos por iteración, proyecto final

13. Conclusiones y recomendaciones

Se lograron realizar todos los objetivos propuestos al inicio de la tesis. Se demostró que el modelo propuesto es válido para la correcta medición de la productividad de todo el ciclo de desarrollo de software, cuando se utiliza Kanban de Agile. El modelo define todos los actores que están involucrados y cómo medir su productividad. Define a su vez, cuándo se deben tomar las medidas y en qué etapa se debe medir. Además, da una categorización por tipo de día ocioso que se puede utilizar para otras valoraciones. Este modelo da todas las herramientas necesarias para hacer una medición de la productividad completa y da las medidas necesarias para poder hacer comparaciones entre equipos de trabajo y proyectos. Finalmente se implementó el modelo en 3 proyectos que probaron su aplicabilidad y validez.

Al tener una tabla de control con todos los datos bien documentados, se pueden hacer muchos tipos de análisis diferentes. Por ejemplo, al tener un grupo ya maduro con varias iteraciones realizadas y diferentes proyectos medidos, se pueden hacer comparaciones para ver cuáles etapas son las más problemáticas o cuáles actores son los que generan más días. Esto va a ayudar mucho para enfocar las áreas de mejora y concentrar mayor entrenamiento a los recursos que están fallando.

Se concluyó también que no hay relación entre las iteraciones y la productividad. Por lo que al comparar la primera iteración con las últimas, por ejemplo, no se puede decir que las primeras sean las de menor productividad. Tampoco existen patrones visibles que relacionen las etapas y actores con la productividad obtenida. Cada iteración va a ser independiente de las demás y cada una tiene que ser evaluada bien para buscar puntos a mejorar.

Una conclusión a la que también se llegó, es que cuando se da una situación que amerita agregar un día ocioso, se tiene que monitorear bien, ya que es muy posible que al darse una repercuta en otras y se generen más días ociosos. Cada situación no termina con sólo marcar el día, se le debe dar seguimiento y hacer todo lo posible por contrarrestar posibles problemas subsecuentes.

El modelo propone utilizar "día ocioso" como unidad de medición, ya que facilita la comparación entre proyectos y grupos de trabajo. El utilizar "hora" no está descartado, pero se complicaría la comparación entre equipos de trabajo que tengan diferentes horarios, como por ejemplo, cuando un equipo trabaje 12 horas al día. En caso que se quiera utilizar "hora" para tener una mayor granularidad de los datos, es importante que los proyectos que se quieran comparar sean medidos de la misma forma desde el inicio, porque si al final del proyecto se pasan las horas a días, se tiene que tomar en cuenta el horario laboral y la cantidad de horas que trabajó cada recurso, lo que dificultaría la comparación y podría llevar a errores en los cálculos.

El modelo propuesto define una escala de medición que se concluyó al realizar varias comparaciones, esto no significa que todos los proyectos y equipos tengan la misma escala, esto se debe definir basado en la productividad aceptable según el equipo. Pero se recomienda que si se pretenden hacer comparaciones entre los proyectos y equipos, debe utilizarse la misma escala.

El modelo también describe una serie de situaciones que se recopilaron en la investigación, pero no significa que sean las únicas que existen. Dado que el tiempo de la tesis fue corto y se utilizaron proyectos de corta duración, es posible que hayan más variantes de días

ociosos. Por esto se recomienda a todos los miembros de los grupos de desarrollo, estar atentos a todas las situaciones que se puedan presentar que generen días ociosos y documentar cada situación detalladamente, y de ser necesario, utilizar otras categorías que faciliten su estudio y evaluación posteriores.

Podría suceder que alguno de los actores quiera esconder alguna situación, para evitar que se registre un día ocioso. Esto sólo va a repercutir de forma negativa, ya que si la productividad no se midió correctamente y la situación no se documentó, es muy posible que el problema no se llegue a solucionar y pueda darse de nuevo. Esto únicamente generaría más retrasos y pérdida de los recursos. Por esto se recomienda nunca descartar u omitir alguna situación que genere días ociosos, para poder mitigar todos los problemas lo antes posible y de la mejor manera posible.

Finalmente, se concluyó que este modelo no se aplica sólo a procesos de desarrollo de software. El modelo descrito se puede aplicar a otros procesos donde es importante medir la productividad y se puedan dar situaciones que generen días ociosos. Es cuestión de hacer una investigación de los diferentes problemas que pueden afectar la producción y llevar un control de los días que se van perdiendo. El resto del modelo se aplicaría de la misma forma descrita en esta tesis, al igual que todos los cálculos que aquí se describieron para obtener la productividad real.

14. Referencias bibliográficas

Adkins, L. (2010) *Coaching Agile Teams: A Companion for ScrumMasters, Agile Coaches, and Project Managers in Transition.* Addison-Wesley Professional.

Anderson, D. J. (2010) *Kanban: Successful Evolutionary Change for Your Technology Business.* Blue Hole Press.

Bahli, B., & Büyükkurt, M. D. (2005) *Group Performance in Information Systems Project Groups: An Empirical Study. Journal of Information Technology Education, 4*, 97-113.

Beck, K., Beedle, M., Martin, R. C., Grenning, J., & Highsmith, J. (2013) *Agile Manifesto.* Recuperado el 13 de Setiembre de 2013, de Agile Manifesto: http://agilemanifesto.org/

Card, D. N. (2006) *The Challenge of Productivity Measurement. Pacific Northwest Software Quality Conference.* Obtenido de http://www.netvalence.net/sites/default/files/resources/recom_read/TheChallengeofProductivityMeasurement.pdf

Cohn, M. (2004) *User Stories Applied: For Agile Software Development.* Addison-Wesley Professional.

Cohn, M. (2005) *Agile Estimating and Planning.* Prentice Hall.

Derby, E., & Larsen, D. (2013) *Agile Retrospectives: Making Good Teams Great!* Pragmatic Bookshelf.

El Emam, K., & Card, D. (2002) *ISO/IEC Standard 15939: Software Measurement Process, International Organization for Standardization.*

Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. (1999) *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Professional.

Galen, R. (209). *Scrum Product Ownership -- Balancing Value from the Inside Out.* RGCG, LLC.

IEEE. (1993) *IEEE Xplore Digital Library.* Obtenido de http://ieeexplore.ieee.org/servlet/opac?punumber=2858

Kan, S. H. (2002) *Metrics and Models in Software Quality Engineering.* Addison Wesley.

Kitchenham, B., & Mendes, E. (2004) Software Productivity Measurement Using Multiple Size Measures. *Software Engineering, IEEE Transactions on, 30*(12), 1023-1035.

Na, K.-S., Simpson, J. T., Li, X., Singh, T., & Kim, K.-Y. (2007) *Software development risk and project performance measurement: Evidence in Korea. The Journal of Systems and Software 80*, 596-605.

Numrich, R. W., Hochstein, L., & Basili, V. R. (2005) *A Metric Space for Productivity Measurement in Software Development. Workshop on Software engineering for high performance computing applications*, 13-16. Obtenido de http://dl.acm.org/citation.cfm?id=1145319.1145324

Ryman, A., & Reddy, A. (2009) *Software development and delivery performance measurement and management: Optimizing business value in software.* IBM. Obtenido de https://jazz.net/library/content/articles/insight/performance-management.pdf

Schwaber, K., & Sutherland, J. (2013) *The Scrum Guide.*

Spencer, K. F. (2013) *A to XP: The Agile ABC Book.* Agile Kindergarten.

Sudhakar, G. P. (2010) *Understanding Software Development Team Performance. Scientific Annals of the'Alexandru Ioan Cuza'University of Iasi: Economic Sciences Series*, 505-513. Obtenido de http://ssrn.com/abstract=2425395

Sudhakar, G. P., Farooq, A., & Patnaik, S. (2012) *Measuring Productivity of Software Development Teams. Serbian Journal of Management, 7*(1), 65-75.

Wallace, L., Keil, M., & Rai, A. (2004) *How Software Project Risk affects Project Performance: An Investigation of the Dimensions of Risk and an Exploratory Model. Decision Sciences, 35*(2), 289-321.

15. Apéndices y Anexos

Abstract

How many times have we wonder if our software development team is efficient or even if it is good at software development? A lot of development groups are adopting Agile to improve their processes and be more productive. The problem is that the use of Agile helps improve the processes but it does not indicate how to measure the productivity of the work being done, so, we do not have a way to know if we improved by implementing it. It is

really difficult to determine if a development team is productive since there are a lot of variables to take into consideration and multiple persons involved. In the following thesis we will define a model to measure the productivity of the development groups that are using this Agile, specifically for the ones implementing Kanban. The model takes into consideration all aspects of the software development life cycle using Agile, from the different stages of the process to the different persons involved in it. The model uses the concept of “idle days” to measure productivity, this will indicate how many real days the team was not working or were lost. It also helps identify where the process has problems. All this will facilitate the identification of improvement points in the process and inside the development team. It will also give us a point of reference to compare different projects and teams, helping us know if we are using the resources in the best possible way.

Keywords: Agile, Kanban, Metrics, Productivity, Software Development