UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE CIENCIAS QUÍMICAS Y FARMACIA

ESTUDIO SOBRE LA ADMINISTRACIÓN EFECTIVA DE PROYECTOS DE SOFTWARE, UTILIZANDO LA METODOLOGÍA SCRUM, DIRIGIDO A EMPRESAS DE SISTEMAS

Pablo César Paniagua González

Maestría en Administración Industrial y de Empresas de Servicios

Guatemala, noviembre de 2017

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE CIENCIAS QUÍMICAS Y FARMACIA

ESTUDIO SOBRE LA ADMINISTRACIÓN EFECTIVA DE PROYECTOS DE SOFTWARE, UTILIZANDO LA METODOLOGÍA SCRUM, DIRIGIDO A EMPRESAS DE SISTEMAS

Trabajo de graduación presentado por

Pablo César Paniagua González

Para optar al grado de Maestro en Artes

Maestría en Administración Industrial y de Empresas de Servicios

Guatemala, noviembre de 2017

JUNTA DIRECTIVA

FACULTAD DE CIENCIAS QUÍMICAS Y FARMACIA

Dr. Rubén Dariel Velásquez Miranda	Decano
M.A. Elsa Julieta Salazar de Ariza	Secretaria
MSc. Miriam Carolina Guzmán Quilo	Vocal I
Dr. Juan Francisco Pérez Sabino	Vocal II
Lic. Carlos Manuel Maldonado Aguilera	Vocal III
BR. Andreína Delia Irene López Hernández	Vocal IV
BR. Carol Andrea Betancourt Herrera	Vocal V

CONSEJO ACADÉMICO

ESCUELA DE ESTUDIOS DE POSTGRADO

Rubén Dariel Velásquez Miranda, Ph.D.
María Ernestina Ardón Quezada, MSc.
Jorge Mario Gómez Castillo, MA.
Clara Aurora García González, MA.
José Estuardo López Coronado, MA.

ACTO QUE DEDICO A:

Mi querida mamá Blanca Lidia González Hernández†:
Tu amor a la vida me hizo contemplar de mejor forma cada momento. Gracias por ayudarme en cada instante de mi vida. Te recuerdo siempre. ¡Infinitamente agradecido mamá!

Mi papá Juan Pablo Paniagua García:
Muchas gracias por inspirarme y motivarme siempre con tu ejemplo.

Mi tío Mario Leonel González Hernández†:
Eternamente agradecido por tus inmensos actos de generosidad.

Mi tía Delia Paniagua García†:
Mil gracias por tu ayuda y compañía en los momentos más difíciles.

"Da siempre lo mejor… y lo mejor vendrá"
Santa Madre Teresa de Calcuta

AGRADECIMIENTOS A:

Dios:
Tu Divina Misericordia y Providencia permiten que pueda concluir con esta etapa. Muchas gracias por acompañarme en cada instante de mi vida.

Virgen María:
Reina y Madre de Misericordia. Gracias por tu intercesión y por acercarnos más a Jesús.

Padres Lidia González y Juan Pablo Paniagua:
Sus esfuerzos y sacrificios son los pilares que lograron que hoy concluya con este ciclo; ¡quedo eternamente agradecido! y quiero que sientan que este logro es también de ustedes.

Hermanas Lilian Paniagua y Mireya Paniagua:
Su fraternidad y perseverancia en la vida me motivan a ser una mejor persona. ¡Gracias por ser un ejemplo de vida!

Amigos:
Su experiencia y conocimiento me permitió crecer tanto personal, profesional y académicamente. Gracias por su amistad y el aprendizaje compartido con todos los compañeros de la MAIES 2015 – 2016, especialmente a Diego Bedoya, Duglas Ruano, Lesbia Guerra, MariaJosé Hesse, Renato Valdez, René Cano y Vivian González.

Catedráticos MAIES 2015 – 2016:
Gracias por compartir su valioso conocimiento.

RESUMEN EJECUTIVO

La metodología Scrum, un tipo de desarrollo de software, permite la administración efectiva de proyectos a través de un conjunto de buenas prácticas que fomentan el trabajo en equipo; tiene su origen en el desempeño de equipos altamente productivos y en el juego de rugby en donde el equipo actúa en común.

Con esta metodología se realiza el desarrollo del sistema entregándolo en fases incrementales, lo cual permite al cliente y al equipo técnico colaborar efectivamente en el proyecto; simplifica trámites burocráticos lo que optimiza el tiempo de gestión. Dividir el desarrollo del sistema por fases simplifica las tareas de instalación, contrario a lo que sucedería si se realizará todo el sistema en una sola entrega.

En Scrum no hay jerarquías especificas sino más bien un equipo auto gestionado que trabaja con el fin de obtener un software de calidad; si bien, es cierto que hay tres roles, estos no son jerárquicos sino más bien de trabajo en equipo. Los roles son dueño del producto quien es el representante del cliente, el equipo quien es el encargado técnicamente de llevar a cabo el proyecto y el desarrollo de dicho software, ayuda a resolver efectivamente cualquier inconveniente, principalmente administrativo, que surja durante su desarrollo.

Con esta metodología se logra la satisfacción del cliente y calidad del desarrollo a través del trabajo en equipo. Los requerimientos pueden modificarse en el transcurso del proyecto, dada la flexibilidad que dicho software propone; esta metodología, permite que los clientes puedan ver el avance entre cada iteración y asegurarse de que el sistema final cumpla con la calidad esperada.

Este tipo de desarrollo de software está especialmente indicado para equipos que puedan autogestionar proyectos, en entornos complejos con requerimientos cambiantes o poco definidos, en donde se necesita obtener rápidamente resultados de buena calidad.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

I. INTRODUCCIÓN

El desarrollo mundial del software se ha enfocado en el control del proceso mediante una rigurosa definición de actividades como el modelado, planificación y documentación detallada. Este esquema tradicional ha demostrado inefectividad en grandes proyectos; incluso en grandes organizaciones en donde tuvieron malas experiencias, como en el FBI con el proyecto Sentinel y en la NASA con el Sistema de planeación gradual de programas, en los que se invirtió grandes cantidades de recursos y tiempo sin poder obtener el software requerido. Es evidente que los proyectos de software son cambiantes y su desarrollo exige tiempos reducidos y una alta calidad.

El desarrollo de un proyecto de software, esta regularmente enmarcado por una restricción en el tiempo de entrega; realizado por medio de una metodología tradicional como cascada, que implica inflexibilidad en la gestión del proyecto, hace que muchos equipos de desarrollo hagan malas prácticas de la ingeniería del software, con el riesgo que esto conlleva. De tal cuenta, surge la metodología ágil de desarrollo de software; su nombre proviene del juego de rugby y se refiere a la forma en que el equipo se desempeña en común para mover la pelota en la cancha, con acoplamiento en busca del mismo propósito y metas claras.

Scrum es una metodología para la gestión de proyectos de software, que ha sido exitosa en grandes empresas como Google y Amazon, entre otras. Su éxito radica en que acoge incertidumbre, creatividad y trabajo en equipo donde cada integrante es autónomo en sus tareas. El control flexible y la predictibilidad del proyecto es una de las grandes virtudes de esta metodología.

En esta investigación se estudió la agilización del proceso de desarrollo de software por medio de la metodología Scrum; se utilizó como apoyo las diferentes fuentes

bibliográficas como artículos, tesis, videos, sitios de internet y principalmente libros, en busca de orientar a las empresas de sistemas para que la puedan implementar.

Se encontró que Scrum es una metodología innovadora en la que se cambia la forma de gestionar tradicionalmente los proyectos de software. Con esta metodología se aplica un conjunto de buenas prácticas que fomentan el trabajo en equipo y una buena comunicación, en donde la información fluye adecuadamente.

El hecho de que el equipo defina con el cliente la lista de requerimientos de acuerdo a prioridades, provee una estimación más efectiva y el equipo puede aportar ideas al cliente lo cual hace que el sistema sea más funcional y ambas partes estén conscientes de lo acordado.

La entrega en fases del proyecto final es una de las características que hacen de Scrum una metodología flexible que permite gestionar de mejor forma el proyecto ya que al dividirlo en tareas pequeñas se facilita su estimación. La entrega de cada fase se realiza de acuerdo a las prioridades del cliente.

Por medio de la metodología Scrum se entrega a tiempo un proyecto de software, se logra, con la colaboración, buen trabajo en equipo, buena comunicación con el cliente, flexibilidad en la planificación, entregas parciales del software final y la resolución inmediata de problemas que surgen durante el desarrollo del proyecto.

II. ANTECEDENTES

1. Aspectos teóricos

1.1. Sistema

El término sistema es tan diverso que puede referirse a sistemas informáticos, sistemas operativos, sistemas de pago, sistema educacional, sistema de gobierno, entre otros. En este trabajo se estudia el término sistema desde el punto de vista informático, para lo cual es útil la siguiente definición: un sistema es una colección de componentes interrelacionados que trabajan conjuntamente para cumplir algún objetivo. Los sistemas que incluyen software se dividen en dos categorías:

1.1.1. Sistemas técnicos informáticos: incluyen componentes hardware y software, pero no procedimientos y procesos. Por ejemplo, las televisiones, los teléfonos móviles y el software de las computadoras personales. En este sentido, el conocimiento de las personas que utilizan un sistema técnico no forma parte del mismo. Por ejemplo, una hoja de cálculo no es consciente del uso que se dará a las operaciones que se calculan en ella. (Sommerville, 2005).

1.1.2. Sistemas socio-técnicos: están conformados por uno o más sistemas técnicos y por el conocimiento de cómo debe usarse el sistema para alcanzar algún objetivo más amplio. Las personas son una parte inherente en estos sistemas, hay políticas y reglas organizacionales y se pueden ver afectados por restricciones externas como las leyes de un país. Estos sistemas no son deterministas lo cual significa que, para una entrada en específico, no siempre producen la misma salida. Su comportamiento depende

de operadores humanos, queda sujeto a la forma de como la persona haya manipulado el sistema. (Sommerville, 2005).

1.2. Software

El software es algo más que los programas de computadora, e incluye documentos asociados y configuración de datos necesarios para que los programas puedan funcionar de forma correcta. Un software es conformado por varios programas independientes, archivos de configuración que se utilizan para ejecutar estos programas, la documentación que describe la estructura del sistema, la documentación para el usuario que explica cómo utilizarlo y los sitios web en donde encontrar información de productos recientes. Los ingenieros de software se encargan de desarrollar estos productos. Hay dos tipos de software:

1.2.1. Productos genéricos: son sistemas que crea una organización de desarrollo y los ofrece al mercado para quien los desee comprar. Por ejemplo, software como procesadores de texto, hojas de cálculo, paquetes para dibujo, herramientas para gestión de proyectos, entre otros. (Sommerville, 2005).

1.2.2. Productos hechos a la medida: son sistemas que cumplen funciones específicas requeridas por el cliente. Por ejemplo, sistemas bancarios, sistemas médicos, sistemas para telefonía, entre otros.

La diferencia con los genéricos es que la organización tiene el control de las especificaciones mientras que, en el software hecho a la medida, la especificación es controlada por el cliente que requirió el software.

Existe también software que inicialmente es genérico, pero que se puede adaptar a las necesidades del cliente; tal es el caso de los sistemas de planificación de

recursos empresariales (ERP), por ejemplo, SAP, en donde se adecua el software para cubrir las reglas del negocio, procesos, informes y otros. (Sommerville, 2005).

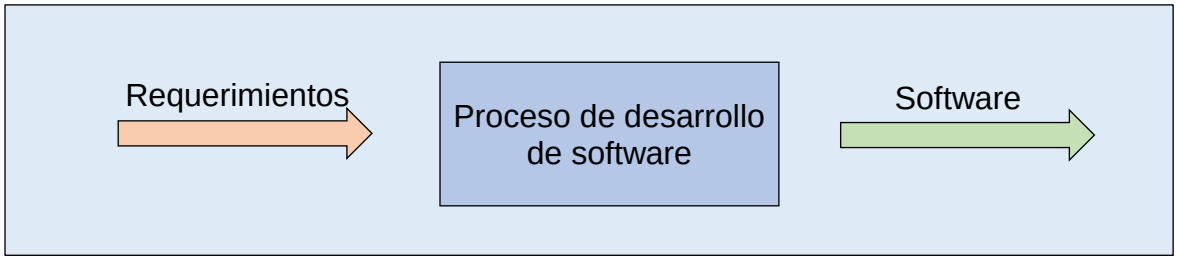
1.3. Proyecto de software

Es el conjunto de actividades destinadas al desarrollo de software y abarca todos sus componentes principales con la finalidad de crear un software de calidad listo para ser utilizado. En la creación del software se involucran diferentes tareas como la toma de requerimientos, análisis y diseño, la gestión del proyecto y su desarrollo. En él intervienen diferentes roles, como lo son: encargado del proyecto, analista, desarrollador, administrador de base de datos, entre otros. (Ramos Cardozzo, 2016).

1.4. Proceso de desarrollo de software

Es una estructura utilizada para el desarrollo de un producto de software, detallado en la figura 1. También es conocido como ciclo de vida del software. Existen diversos modelos que son utilizados para llevar a cabo el desarrollo del software, cada uno con fases propias que se ejecutan durante el progreso del proyecto.

Figura 1. Proceso de desarrollo de software.



Fuente: Elaboración propia.

1.4.1. La toma de requerimientos de los usuarios: se lleva a cabo durante la fase de análisis del proyecto. Muchos de los clientes no saben con claridad que es lo que requieren; por ello, el analista se comunica

constantemente con el cliente para ayudarlo en la definición de sus requerimientos. Durante el análisis, inicialmente se trata de recopilar la mayor cantidad de información posible que proporcione el cliente; luego, se debe interrogar y aclarar dudas acerca de la información obtenida. Después, el analista debe comprobar la información y sugerir soluciones; finalmente, ya que se tiene la información necesaria del problema, se procede a realizar el documento con la especificación de los requerimientos.

1.4.2. Especificación de las características requeridas del sistema: esta, es la última fase del análisis en la cual se recopila información con claridad sobre cuál debe ser el comportamiento del sistema requerido. Para ello, son útiles los documentos formales del negocio y ejemplos del funcionamiento manual de lo que se automatizará.

1.4.3. Crear una solución adecuada al problema por medio del sistema que se diseña: por medio del proyecto se da solución a los requerimientos del cliente y con base en la experiencia acumulada se crea el documento del proyecto con el diseño y planificación del sistema, el cual cumple con técnicas estandarizadas.

1.4.4. Desarrollar la solución propuesta: en esta fase se realiza el desarrollo del software, aquí los programadores escriben el código que dará vida al sistema, se hace la documentación técnica y la documentación para el usuario final, se soluciona cualquier error que se detecte en el transcurso del proyecto, se hacen pruebas del sistema y los módulos que lo integran. (Noriega, 2015).

El objetivo es garantizar que la solución creada corresponda al problema planteado por el cliente y que su funcionamiento sea el que se esperaba.

1.4.5. Modificar las soluciones cuando existen nuevos requerimientos o requerimientos que no se había identificado: las necesidades de quienes requieren el sistema cambian en el transcurso del tiempo; por tanto, es necesario readecuarlo de acuerdo a las nuevas necesidades que se observan cuando el sistema está en funcionamiento y que no fue posible ver con anterioridad a su creación, ya sea porque la necesidad no existía o porque se pasó por alto algún factor importante para incluirlo.

Esto da origen a un cambio en el proyecto ya desarrollado, con un nuevo análisis, desarrollo de las nuevas funcionalidades, pruebas extras y en general trabajo adicional en el proyecto. (Noriega, 2015).

En paralelo, el encargado del proyecto, se debe comunicar con el cliente continuamente para acordar los plazos y las características de la entrega del software. Gestiona todas las actividades, tiene la sincronización del proyecto y se comunica con los involucrados en la creación del sistema.

Para ello, se realiza un cronograma con las tareas de cada persona. Debe ver que se cuente con los recursos necesarios para que no haya retraso en el desarrollo del proyecto y que las tareas puedan alcanzarse objetivamente. También, se evalúa la eficacia de todas las actividades para que el proyecto se finalice en el tiempo estipulado.

2. Modelos de desarrollo de software

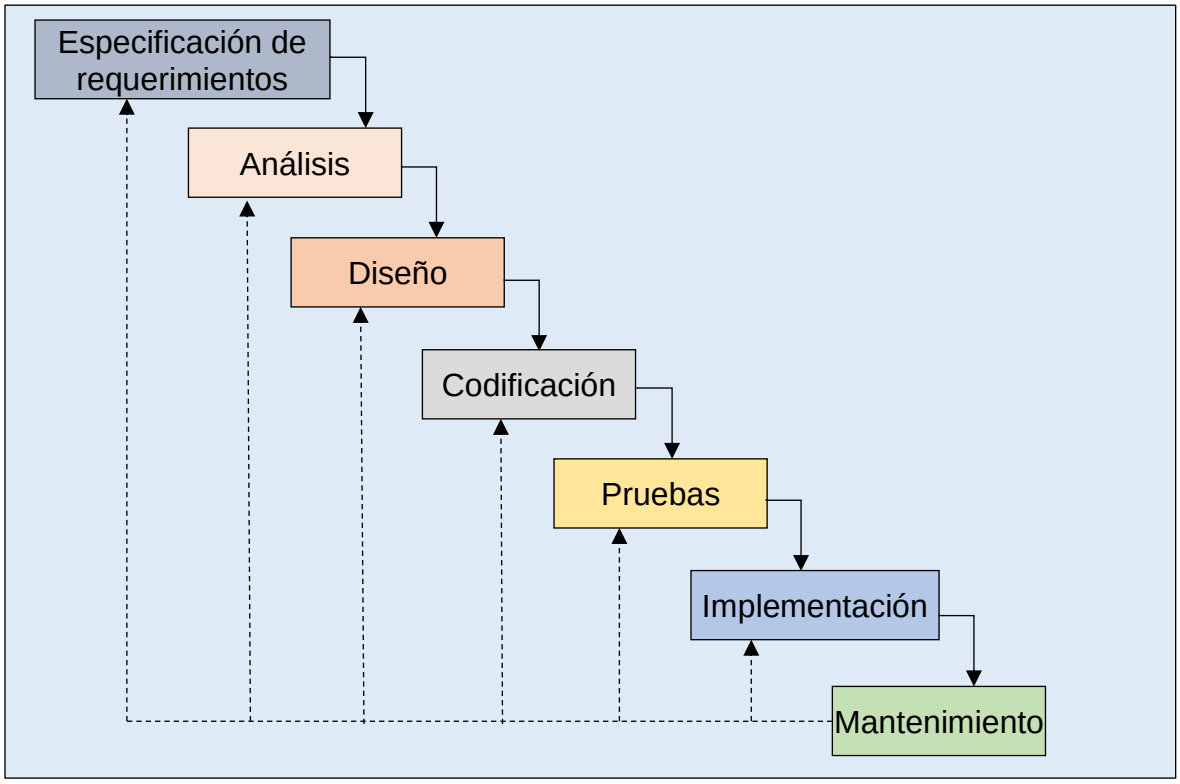
Estos modelos han sido creados con el objetivo de organizar el proceso del desarrollo de software, para obtener un software eficiente que satisfaga los requerimientos del cliente en un tiempo razonable.

A continuación, se describen algunos modelos. (Noriega, 2015).

2.1. Modelo tradicional en cascada

Se conoce como modelo clásico, modelo tradicional o modelo lineal secuencial. Este método es considerado como el enfoque clásico para el ciclo de vida del desarrollo de sistemas, implica un desarrollo rígido y lineal. (Cortés, 2006).

Figura 2. Modelo cascada.



Fuente: Sommerville, 2005. p.62.

Las fases siguientes se llevan a cabo secuencialmente:

2.1.1. Especificación de requerimientos

2.1.2. Análisis

2.1.3. Diseño

2.1.4. Codificación

2.1.5. Pruebas

2.1.6. Implementación

2.1.7. Mantenimiento

Se dirige por documentos y no proporciona resultados tangibles de software hasta que se ha finalizado su desarrollo. Existe una secuencia definida en la que es necesario terminar una fase antes de iniciar con la siguiente, hasta que se obtiene el software final. Es conveniente en proyectos de software que son estables, principalmente cuando sus requerimientos no son cambiantes.

Cuando se hace la revisión del software y se determina que el proyecto no está listo para pasar a la siguiente etapa, permanece en la etapa actual hasta que esté preparado. Este modelo tiene forma de cascada de agua con varios saltos, como se muestra en la figura 2, en la que cada salto representa cada una de las fases del ciclo de vida. A grandes rasgos, se inicia con la especificación de requerimientos, para luego convertirlo en un documento de análisis y diseño del software, para que este pueda ser codificado; cuando está el software listo, se procede con las pruebas en las que se trata de detectar cualquier error tanto técnico como de funcionalidad, se brindan las correcciones necesarias y se procede a la implementación del producto para que este pueda ser utilizado por el cliente que lo solicitó y se queda en contacto con el cliente para el mantenimiento respectivo del software.

Cada una de las tareas se evalúa por separado y el equipo que lo desarrolla es diferente. Al utilizar esta metodología es necesario hacer un análisis de la situación, por ejemplo, si el cliente quiere intervenir en el proceso una vez iniciado, este método no sería el indicado; para ello sería necesario un método iterativo. La especificación de requerimientos no puede ser modificada después de que se ha concluido con esta etapa. Una modificación durante la ejecución de alguna de las fases, implicaría reiniciar desde el principio todo el ciclo completo, lo cual implicaría mayor inversión de tiempo y desarrollo. (Cortés, 2006).

El modelo en cascada tiene un enfoque estructurado y progresa linealmente a través de sus fases, por lo que resulta fácil de entender. Este proceso se utiliza frecuentemente en los proyectos de gobierno y en proyectos que requieren poca innovación, también en software de pequeño y mediano tamaño. Hay variantes de

este modelo que se conoce como modelo cascada realimentado, en el que es permitido dar sugerencias entre etapas, permite realizar cambios entre cada etapa, y de esta forma se puede retroceder de una etapa a la anterior o incluso saltar a otras anteriores si es requerido. En el modelo cascada y cascada realimentado no se tiene en cuenta la naturaleza evolutiva del software y se planifica como estático con requerimientos bien conocidos y definidos desde el inicio. (Cortés, 2006).

2.2. Ventajas del modelo cascada

- Permite la departamentalización y control de gestión.
- El horario se establece con los plazos normalmente adecuados para cada etapa de desarrollo.
- La gestión de proyectos se facilita con este modelo.
- Se tiene un control detallado del proyecto.
- Limita la cantidad de interacción entre equipos que se produce durante el desarrollo.

2.3. Desventajas del modelo cascada

- No se conoce si la solución es correcta hasta estar cerca de su entrega.
- Hay tiempo para corregir fallas.
- Los cambios introducidos durante el desarrollo pueden confundir al equipo profesional en las etapas tempranas del proyecto.
- El proceso es lento.
- Un cambio implica un gran esfuerzo para llevarlo a cabo.

Cuando la aplicación ya está en fase de prueba y es necesario cambiar algo que no se pensó en las primeras etapas, sería mejor usar un método diferente al de cascada; por ejemplo, el modelo en espiral.

En la práctica el modelo en cascada no cumple las expectativas, ya que por lo general el cliente pide cambios ya iniciado el proceso, y solicita ver los avances del desarrollo; esto hace que este modelo casi no se utilice.

Las actividades están estrictamente ligadas, esto hace que una actividad no empiece hasta que se termina totalmente la anterior, lo que a veces se traduce en tiempo perdido al tener que esperar la evolución de cada una de las etapas.

Está basado en el ciclo convencional de una ingeniería y su visión es simple, el desarrollo del software se debe realizar una secuencia de fases. (Cortés, 2006).

2.4. Modelo espiral

El modelo espiral que definido por Boehm en 1988 como un modelo de proceso de software evolutivo, utiliza las mejores características del ciclo de vida clásico, creación de prototipos y agrega un nuevo elemento: análisis de riesgo.

Este modelo propone el desarrollo rápido de versiones incrementales del software y entrega en cada iteración, una versión más completa hasta llegar a la versión final. El modelo espiral de la figura 3 define cuatro actividades principales, también llamadas regiones de tareas o sectores:

2.4.1. Planificación: durante la primera vuelta alrededor de la espiral se definen los objetivos, las alternativas y las restricciones, se analizan e identifican los riesgos.

Si en el análisis de riesgo se encuentran dudas sobre los requerimientos, se puede usar la creación de prototipos en el cuadrante de ingeniería para solucionar las dudas del cliente y el equipo de desarrollo.

2.4.2. Análisis de riesgo: se hace una revisión del proyecto y se ve si es necesario continuar con un ciclo posterior de la espiral. Si se decide continuar, se realizan los planes para la siguiente fase del proyecto.

2.4.3. Ingeniería: en este sector se desarrolla y se valida. Después de evaluar los riesgos, se elige un modelo para el desarrollo del sistema.

2.4.4. Evaluación del cliente: el cliente evalúa el trabajo de ingeniería y sugiere modificaciones. En base a los comentarios del cliente se genera la siguiente fase de planificación y análisis de riesgo.

En cada bucle alrededor de la espiral, la culminación del análisis de riesgo resulta en una decisión de "seguir o no seguir".

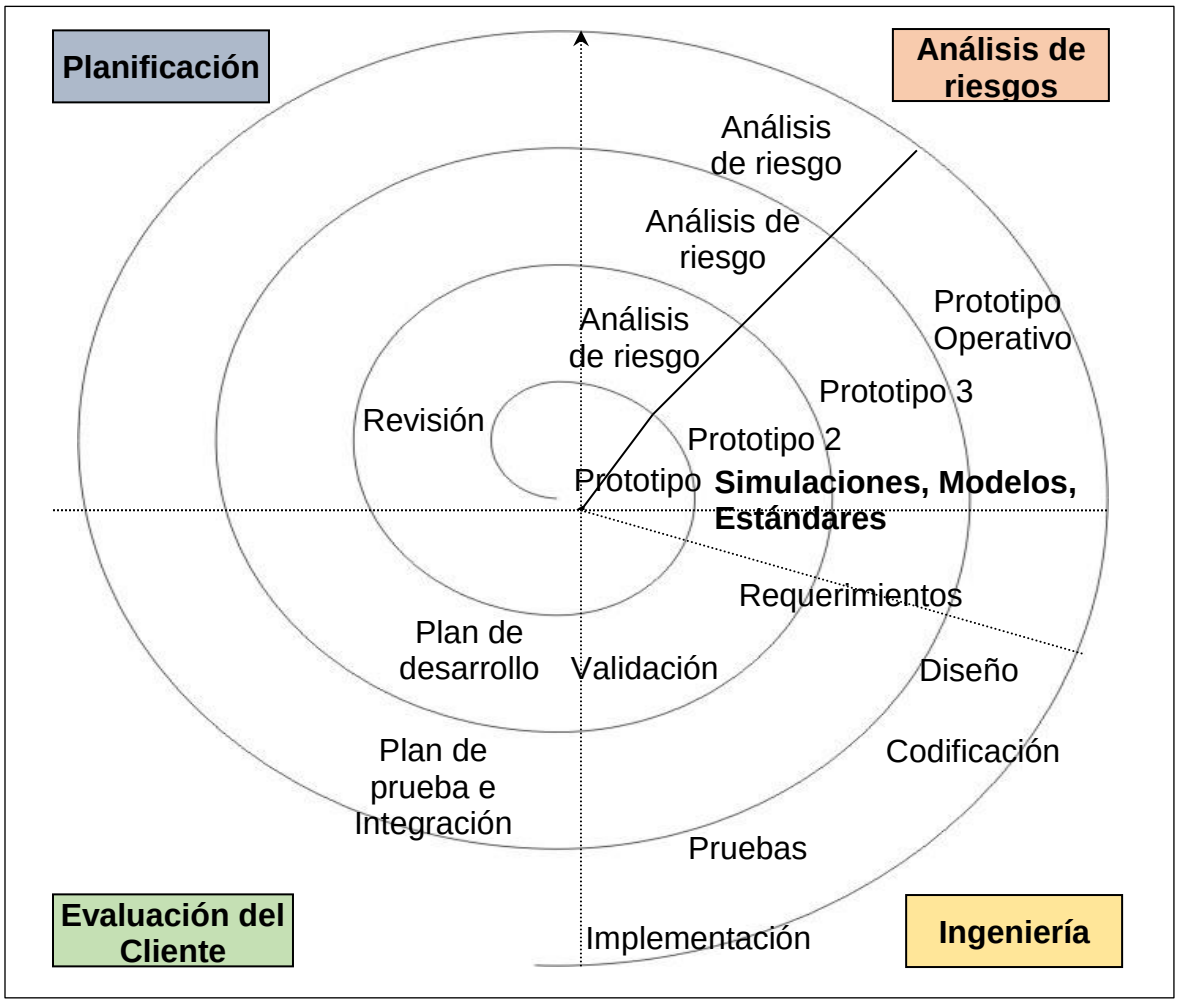
Con cada iteración alrededor de la espiral, se inicia en el centro y sigue hacia el exterior, se construyen sucesivas versiones del software, cada vez más completas y, al final, el propio sistema operacional. (Pressman, 2005).

Un ciclo en espiral inicia con la elaboración de objetivos, rendimiento y funcionalidad. Después se describen las alternativas para alcanzar dichos objetivos y las restricciones posibles en cada una de ellas. Cada alternativa se evalúa contra cada objetivo y se identifican las fuentes de riesgo del proyecto.

El siguiente paso es resolver los riesgos por medio de hacer un análisis más detallado como la construcción de prototipos y la simulación. Ya que se evaluaron los riesgos, se lleva a cabo el desarrollo, seguido de una actividad de planificación para la siguiente fase. Esto permite que el equipo de desarrollo y el cliente puedan reaccionar a los riesgos en cada nivel.

Utiliza la creación de prototipos en cualquier etapa como un mecanismo de reducción de riesgo. (Sommerville, 2005).

Figura 3. Modelo de Espiral de Boehm.



Fuente: Sommerville, 2005. p.68.

2.5. Modelo incremental

El modelo incremental propone secuencias lineales de forma escalonada durante el avance del proyecto, cada secuencia lineal produce un incremento del software. En el flujo del proceso de cualquier incremento se puede utilizar la construcción de prototipos.

El modelo incremental entrega el software en partes pequeñas, pero utilizables, llamadas “incrementos”. En general, cada incremento se construye sobre aquel que ya ha sido entregado. (Pressman, 2005).
Cuando se utiliza un modelo incremental, el primer incremento a menudo es un producto esencial, en el que se desarrollan los requerimientos básicos. El cliente utiliza este producto para realizar observaciones y se desarrolla un plan para el incremento siguiente. Este proceso se repite con la entrega de cada incremento, hasta que se termina el software. (Pressman, 2005).

El modelo incremental, como la construcción de prototipos y otros enfoques evolutivos, son iterativos por naturaleza. A diferencia de la construcción de prototipos, el modelo incremental se centra en la entrega de un producto operacional con cada incremento. Los primeros incrementos son versiones incompletas del producto final, pero proporcionan al usuario una parte de la funcionalidad y una forma de evaluar el software.

El desarrollo incremental es particularmente útil cuando el personal no está disponible para una implementación completa en la fecha límite que se ha establecido para el proyecto. Los primeros incrementos se pueden implementar con menos personas.

Este modelo constituyó un avance sobre el modelo en cascada, pero también presenta problemas. Aunque permite el cambio continuo de requisitos, aún existe el problema de determinar si los requerimientos propuestos son válidos. Los errores en los requerimientos se presentan tarde y su corrección resulta tan costosa como en el modelo en cascada. (Pressman, 2005).

2.6. Proceso de desarrollo unificado

Es un modelo complejo con mucha terminología propia, pensado principalmente para el desarrollo de grandes proyectos. Es un proceso que puede adaptarse y

extenderse en función de las necesidades de cada empresa. Fue creado por Ivar Jacobson, Grady Booch y James Rumbaugh.

2.7. Características del proceso de desarrollo unificado

El proceso unificado guía a los equipos de proyecto en cómo administrar el desarrollo iterativo de un modo controlado mientras se balancean los requerimientos del negocio, el tiempo al mercado y los riesgos del proyecto. En este modelo se describen los requerimientos y el diseño de la arquitectura del software, y se prueba el sistema de acuerdo a lo solicitado. El proceso describe cómo desarrollar lo que se va a entregar y también patrones de diseño. El proceso unificado es soportado por herramientas que automatizan, entre otras cosas, el modelado visual, la administración de cambios y las pruebas.

El proceso unificado está basado en componentes interconectados a través de interfaces bien definidas. Además, el Proceso Unificado utiliza el UML para expresar gráficamente todos los esquemas de un sistema de software.

Pero, realmente, las características que definen este Proceso Unificado son tres: iterativo e Incremental, Dirigido por casos de uso y Centrado en la Arquitectura. (Jacobson, 2000).

2.8. Dirigido por casos de uso

Un caso de uso es un fragmento de la funcionalidad del sistema que da un resultado de valor a un usuario. Los casos de uso modelan los requerimientos funcionales del sistema.

En base a los casos de uso, los analistas crean una serie de modelos de diseño e implementación que se llevarán a cabo. Estos modelos deben coincidir con los casos de uso. Finalmente, los casos de uso se usan para realizar los casos de

pruebas del software desarrollado. Los casos de uso ayudan a iniciar de mejor forma el proceso de desarrollo y proporcionan un hilo conductor durante el ciclo de vida del desarrollo de software. (Torossi, 2005).
Los casos de uso son utilizados para:

- Establecer el comportamiento deseado del sistema.
- Verificar y validar la arquitectura del sistema.
- Hacer Pruebas.
- Tener una comunicación entre los participantes del proyecto. (Torossi, 2005).

2.9. Centrado en la arquitectura

Arquitectura es el conjunto de decisiones significativas acerca de la organización de un software, la selección de los elementos estructurales a partir de los cuales se compone el sistema, las interfaces entre ellos, comportamiento y composición.

El concepto de arquitectura software abarca los aspectos estáticos y dinámicos más significativos del sistema. Es una vista del diseño completo y resalta sus características principales.

Los casos de uso y la arquitectura están muy relacionados; los casos de uso deben coincidir con la arquitectura y, a su vez, la arquitectura debe permitir el desarrollo de los casos de uso requeridos. El arquitecto desarrolla la forma o arquitectura a partir de la comprensión de un conjunto de los principales casos de uso.

Durante el proceso de arquitectura se crea un esquema en borrador de la arquitectura se inicia por la parte no específica de los casos de uso, como la plataforma, pero con una comprensión general de los casos de uso. Luego se trabaja con un conjunto de casos de uso claves. (Torossi, 2005).

Cada caso de uso es especificado detalladamente y realizado en términos de subsistemas, clases, y componentes. A medida que los casos de uso se especifican se descubre más de la arquitectura que conlleva al diseño de más casos de uso. Este proceso continúa hasta que la arquitectura es estable. (Torossi, 2005).

En la figura 4 se describen las diferencias entre la metodología ágil y la tradicional.

3. Desarrollo ágil

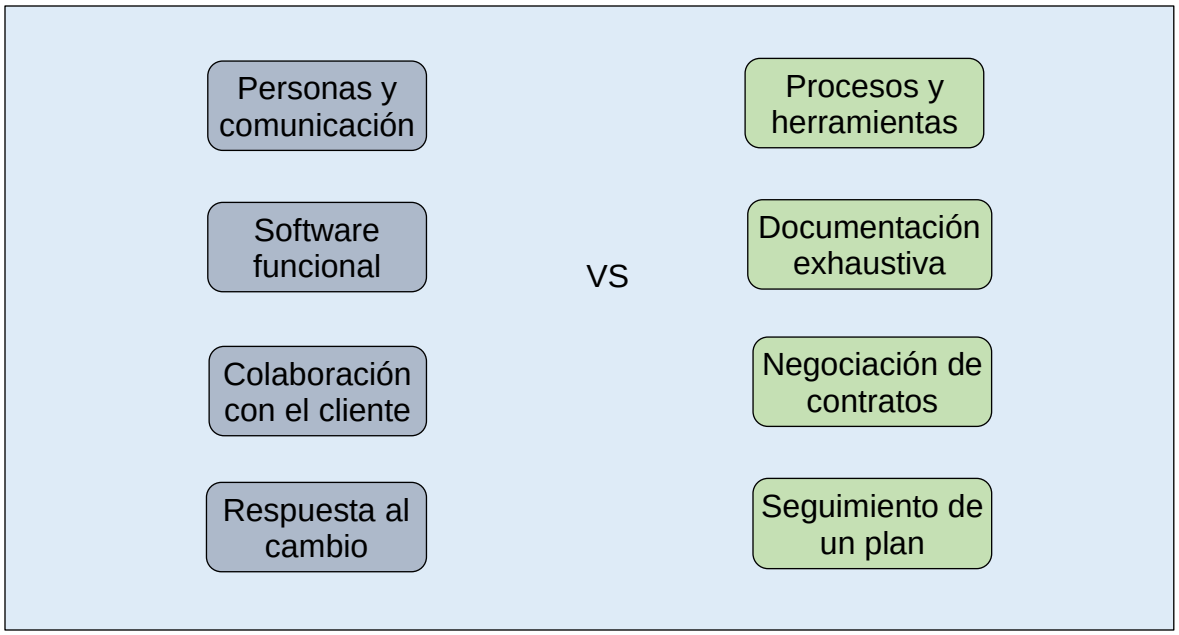
El desarrollo ágil del software surge a mediados de la década de los años 90's con puntos de vista contrarios a los métodos tradicionales como el modelo de desarrollo en cascada, el cual es burocrático y lento para la creación del software.

El desarrollo ágil es una filosofía que describe formas de pensar y trabajar efectiva y ágilmente un proyecto de software. Es un proceso de desarrollo incremental en el que se reducen los riesgos por medio de iteraciones cortas, que conlleva necesariamente a eliminar trámites burocráticos que permitan, de esta forma, rapidez en el desarrollo y entrega del software.

Cada iteración incluye planificación, requerimientos, análisis, diseño, codificación, pruebas y documentación; pero de una forma resumida con respecto de los métodos tradicionales. Al final de cada iteración se entrega un software completamente funcional y sin errores. Al final de la iteración se hace una revisión y se asignan prioridades para el desarrollo siguiente.

Existe una buena comunicación entre todas las partes involucradas; esto incluye a los analistas, codificadores, jefes de proyecto, personas encargadas de hacer pruebas y clientes. Se solucionan las dudas que surgen y se eliminan los errores rápidamente; esto, permite un buen grado de sinergia entre el equipo de desarrollo y alta efectividad en el software. (Pressman, 2010).

Figura 4. Metodología ágil vs. Tradicional.



Fuente: Olivo, 2014. p.75.

3.1. Manifiesto ágil

El término “ágil”, aplicado al desarrollo del software, surge en una convención realizada del 11 al 13 de febrero del 2001, en donde se reunieron 17 expertos de la industria del software, en Utah EEUU. El objetivo primordial de la reunión era tratar las necesidades con que se encontraban las empresas en repetidas ocasiones, en las que había que desarrollar alguna aplicación, más rápido que de costumbre, sin cumplir con toda la formalidad necesaria de una metodología tradicional. Muchas veces, esto ocasionaba un caos en las instituciones ya que no existían parámetros en los que se pudieran guiar para lograr una entrega rápida del software. Es así, como expertos del software vieron la necesidad de reunirse para acordar reglas claras a la hora de desarrollar software rápidamente.

Después de realizada dicha reunión, se crea la conocida Alianza Ágil, una organización sin ánimo de lucro, enfocada en dar ayuda en la implementación del desarrollo ágil a organizaciones. (Cunningham, 2001).

Como punto de partida, la Alianza Ágil crea un documento, en el que se incorpora principios y valores del desarrollo ágil. Este documento, conocido por tener la filosofía ágil, es el llamado Manifiesto Ágil. (Cunningham, 2001).

3.2. Valores del manifiesto ágil

3.2.1. Valorar más a las personas y su interacción que a los procesos y herramientas: las personas son la parte primordial para tener éxito en un proyecto de software. Es mejor formar bien el equipo de personas involucradas en el proyecto y, luego, que sean ellas quienes creen el entorno de desarrollo conforme a sus necesidades y no a la inversa, como suele suceder equívocamente.

3.2.2. Valorar más el desarrollo del software funcional que la documentación extensa: los documentos sirven para dar soporte al software, permiten la transferencia del conocimiento, registran información histórica y son necesarios para trámites legales. La documentación es necesaria; sin embargo, el manifiesto resalta que es menos importante que el software funcional. Los documentos no generan el valor que se logra con la comunicación directa entre las personas y a través de la interacción con los prototipos. Por ello, el manifiesto recomienda reducir al mínimo indispensable el uso de documentación.

3.2.3. Valorar más la colaboración del cliente que la negociación de contratos: es muy importante que exista mucha interacción entre el cliente y el equipo de desarrollo, para garantizar el desarrollo adecuado del proyecto y éxito del mismo. Un contrato no aporta valor al producto, es una formalidad que establece líneas divisorias entre responsabilidades, pude ser utilizado incluso para posibles disputas contractuales entre cliente y proveedor. En el desarrollo ágil el

cliente es un miembro más del equipo, que se integra y colabora en el grupo de trabajo, por lo que se asume que no deberían existir este tipo de formalidades.

3.2.4. Valorar más la respuesta al cambio que el seguimiento de un plan: tiene que existir la habilidad para responder a los cambios que ocurran durante el desarrollo del proyecto.

Puede ser que cambien los requerimientos, la tecnología, el equipo de personas, entre otras, también es importante que la planificación sea flexible y abierta a cambios. (Canós, Letelier, & Penadés, 2009) (Agile Alliance, 2015)

3.3. Principios del Manifiesto Ágil

Los valores descritos anteriormente fueron la base para proponer los principios del Manifiesto Ágil, los cuales hacen la diferencia entre un proceso ágil y uno tradicional. A continuación, se describen los doce principios. Los primeros dos tienen un enfoque general mientras que los siguientes están más enfocados en el proceso de desarrollo.

3.3.1. Lo primordial es la satisfacción del cliente a través de entregas tempranas y continuas del software, para darle más valor.

3.3.2. Los requerimientos cambiantes son bien aceptados, aún en una etapa avanzada del desarrollo. Los procesos ágiles utilizan los cambios para dar al cliente una ventaja competitiva.

3.3.3. Entregar con frecuencia software funcional con una periodicidad desde un par de semanas hasta un par de meses, tratar de tener un intervalo preferiblemente corto de tiempo entre cada entrega.

3.3.4. Personas del negocio y los del equipo de desarrollo deben trabajar juntos a lo largo del proyecto.

3.3.5. Desarrollar proyectos con individuos motivados. Debe darse a éstos el ambiente con el apoyo necesario y confiar en que el equipo hará un buen trabajo. Esto implica que cada empleado puede actuar proactivamente, en busca de los mejores resultados.

3.3.6. La conversación cara a cara es el método más eficiente y eficaz para transmitir la información a los integrantes de un equipo de desarrollo y entre ellos mismos.

3.3.7. Un software funcional es el mejor indicador del avance del proyecto.

3.3.8. Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios deben estar en armonía permanente.

3.3.9. La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.

3.3.10. La simplicidad es esencial: el arte de maximizar la cantidad de trabajo no realizado.

3.3.11. De los equipos auto organizados surgen las mejores arquitecturas, requerimientos y diseños.

3.3.12. Cada cierto tiempo, el equipo debe analizar respecto a cómo llegar a ser más efectivo y de acuerdo a esto, reubicar su comportamiento. Esto se evalúa en las reuniones de seguimiento que se tienen con el equipo. (Cunningham, 2001).

3.4. Habilidades de las personas en el desarrollo ágil

El desarrollo ágil se centra en las habilidades de las personas, adapta el proceso de acuerdo a las necesidades de los integrantes del equipo. En los integrantes del equipo debe existir cierto número de características clave, como son las siguientes:

3.4.1. Competencia: tener las habilidades y el conocimiento relacionado con el software; esto, debe considerarse para todas las personas que son miembros ágiles del equipo.

3.4.2. Enfoque común: los integrantes deben enfocarse en un objetivo común, aunque cada quien realice diferentes tareas. Se debe entregar al cliente, en la fecha prometida, un módulo del software funcional.

3.4.3. Colaboración: la ingeniería de software evalúa, analiza y utiliza la información que se comunica al equipo de software; crea la información que ayudará a los participantes a entender el trabajo del equipo; y genera información que aporte valor al negocio del cliente.

3.4.4. Habilidad para tomar decisiones: el equipo debe tener libertad para tomar las decisiones que considere oportunas, tal como las relativas a asuntos técnicos del proyecto.

Las lecciones aprendidas de cualquier actividad relacionada con la solución de problemas son muy útiles para las etapas posteriores del proyecto. (Canós, Letelier, & Penadés, 2009).

3.4.5. Confianza y respeto mutuos: el equipo ágil debe ser integrado, con la confianza y el respeto para que se vea reflejado en el avance del

proyecto, ya que si hay compañerismo es mucho más fácil transmitir el conocimiento de una persona a otra, dentro del mismo equipo.

3.4.6. Organización propia: el equipo ágil debe auto organizarse para hacer el trabajo, organizar el proceso que se adapte mejor a su entorno y organizar la planificación del trabajo a fin de entregar a tiempo el módulo respectivo del software. El equipo sirve como su propio gerente. (Canós, Letelier, & Penadés, 2009).

4. Metodologías ágiles de desarrollo de software

Un método de desarrollo de software es un conjunto de actividades que ayudan en el proceso del desarrollo del mismo. En los años 90’s, los problemas con los proyectos y la insatisfacción con las metodologías tradicionales de desarrollo de software llevó a “desarrolladores” a proponer nuevos métodos.

El término de Metodologías Ágiles se hizo popular cuando 17 especialistas en desarrollo de software presentaron los métodos: Programación Extrema (XP), Scrum, Desarrollo impulsado por características, entre otros y establecieron principios comunes.

Las metodologías ágiles varían en sus prácticas y en sus fases pero comparten algunas características, tales como: desarrollo iterativo e incremental, comunicación y reducción de productos.

Estas metodologías ágiles se basan en el Manifiesto Ágil. Entre ellas, existen algunas características similares y hay otras en las que sus diferencias son notables. Cada metodología propone procesos diferentes, pero comparte un conjunto de principios comunes. Algo importante es el trabajo en equipo, que ayuda a que el software se desarrolle en tiempo óptimo.

4.1. Principios comunes de las metodologías ágiles

4.1.1. Participación del cliente: los clientes participan activamente en el proceso de desarrollo, en el que aportan los requerimientos del sistema, establecen prioridades, y evalúan cada fase entregada.

4.1.2. Entrega incremental: el software es desarrollado en fase en la que, conforme se entrega cada una, abarca más funcionalidades que la anterior, hasta completar el software.

4.1.3. Orientado en las personas y no en procesos: se da mucho énfasis en las habilidades del equipo de desarrollo.

Los miembros del equipo deben tienen la libertad de proponer y contribuir con el proyecto en cualquier fase.

4.1.4. Aceptar cambios: se da por hecho que habrá cambios en los requerimientos; por ello, el proyecto y el sistema se reacomodan a esos cambios.

4.1.5. Mantener la simplicidad: que se entienda el avance del proyecto para todos los involucrados, incluido el cliente que regularmente no tiene los conocimientos técnicos que los desarrolladores poseen.
El código debe hacerse de una forma entendible para que cualquier integrante del equipo pueda hacer las modificaciones que sean necesarias, en cualquier momento. (Camposantos, 2014).

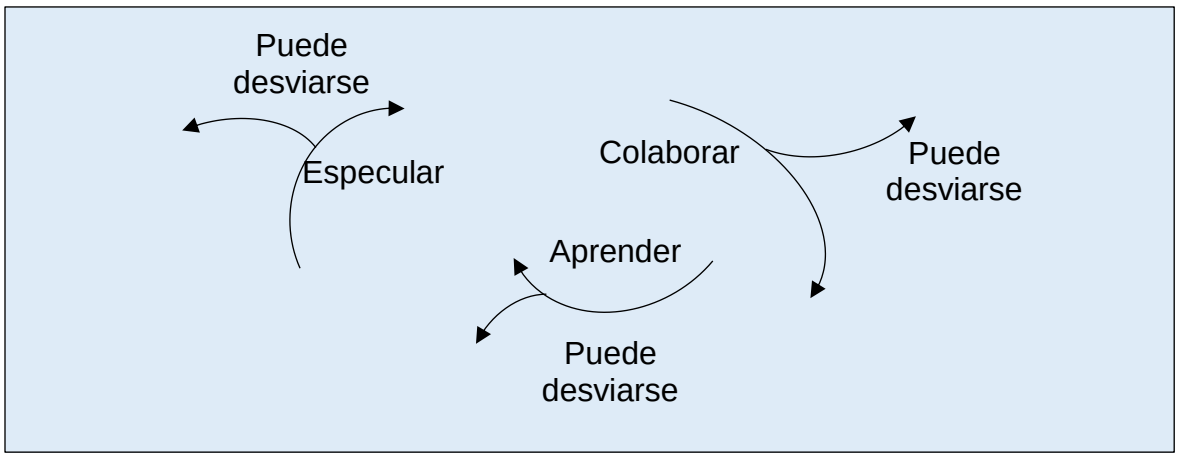
A continuación, se describen brevemente las metodologías agiles más conocidas, a excepción de la metodología Scrum, que es el objetivo del estudio de esta investigación.

4.2. Desarrollo de software adaptativo

Esta metodología, más conocida por su nombre en inglés como “Adaptive Software Development” y sus siglas ASD, fue creada en 1999 por Jim Highsmith. Propone un cambio de filosofía en las organizaciones por medio de la transición del modelo comando-control al modelo liderazgo-colaboración.

Se basa en los conceptos de los sistemas adaptativos complejos que tienen relación con la inteligencia artificial.

Figura 5. El ciclo de vida adaptativo.



Fuente: Highsmith, 2010.

ASD propone utilizar el ciclo de vida de la figura 5: especular, colaborar, aprender. El proyecto inicia con la fase de “especulación” en donde se realiza la planificación del proyecto en función de las entregas; al utilizar la palabra “especular” se ve la naturaleza impredecible de los sistemas complejos. (Highsmith, 2000).

En cada iteración, se aprenderán nuevas funcionalidades y cambiarán los requerimientos. La especulación ASD permite administrar proyectos con alto grado de cambios y en que se necesita un rápido desarrollo.

En esta etapa se utiliza estructura de componentes en orden descendiente, en el que se utiliza un cuadro u hoja de cálculo para describir la funcionalidad a ser

liberada en cada ciclo; sin embargo, no es más que una especulación ya que el carácter adaptativo del proceso permite pequeñas desviaciones en un sentido, por lo que la metodología propone que cada ciclo se componga de una mezcla entre funcionalidades críticas, útiles, y opcionales, y prevenga los posibles retrasos que pudieran surgir o las desviaciones que pueden cambiar el rumbo del proyecto.

La siguiente fase del ciclo de vida: “colaborar”, es aquella en la que se desarrolla la funcionalidad definida durante la especulación. ASD define un componente como un grupo de funcionalidades o “entregables” a ser desarrollados durante un ciclo iterativo.

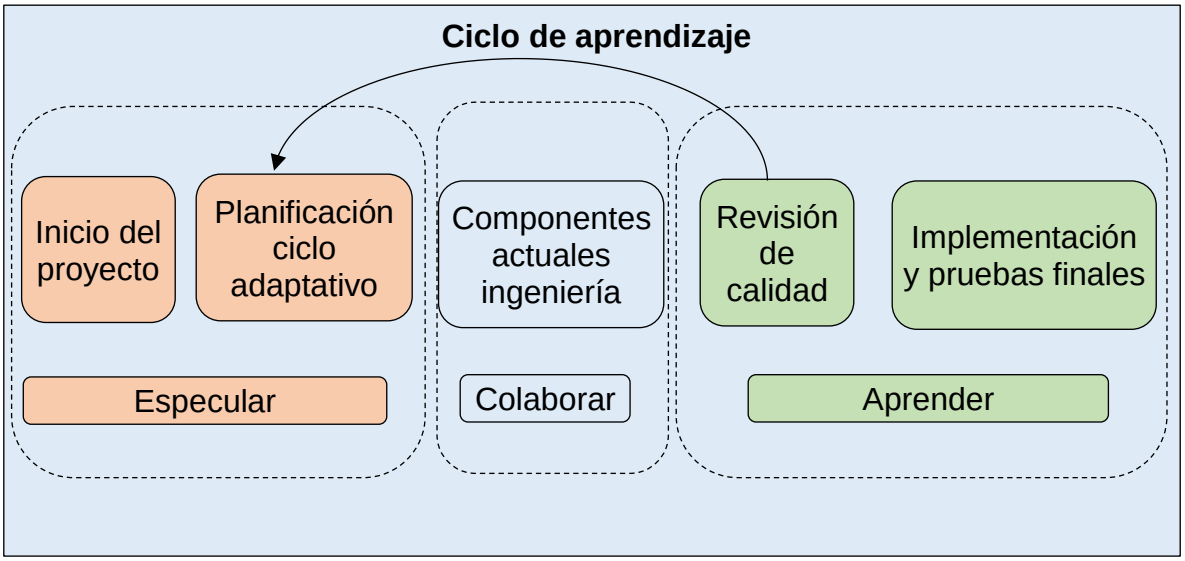
Durante cada iteración el equipo colabora de la mejor forma para entregar la funcionalidad planificada. También, existe la posibilidad de incorporar nuevas alternativas que podrían alterar el rumbo del proyecto. La colaboración entre las personas es imprescindible para la integración del proyecto. La emergencia es una propiedad de los sistemas adaptativos complejos que permite que los grupos de desarrollo trabajen con lo mejor de sí, para trabajar rápidamente. La última fase del ciclo: “aprender”, consiste en la revisión de calidad que se hace al final de cada ciclo, en la que se analizan cuatro categorías de cosas para aprender. (Highsmith, 2000).

- Calidad del software desde la perspectiva del cliente.
- Calidad del software desde la perspectiva técnica.
- El funcionamiento del equipo de desarrollo y las prácticas utilizadas.
- El estado del proyecto.

Para evaluar la calidad desde el punto de vista del cliente se utilizan grupos de enfoque en el cliente, en los que se explora un modelo de la aplicación y se anotan cambios requeridos por el cliente. El enfoque en esta fase, es el de aprender cuales han sido los errores o desvíos y poder resolverlos, sin buscar culpables. Lo importante es que el sistema cumpla con lo que necesita el usuario.

Es importante que el software cumpla con todos los estándares técnicos y una arquitectura adecuada, lo que permitirá que el desempeño del software funcione óptimamente y de esta forma que el cliente este satisfecho con el producto entregado. (Highsmith, 2000).

Figura 6. Actividades del ciclo de vida adaptativo.



Fuente: Highsmith, 2010.

Evaluar la interacción entre los involucrados en el proyecto y la dinámica de grupo, sirve para medir el desempeño y el grado de cohesión del equipo. Esta evaluación se puede realizar al final de cada ciclo por medio de pequeñas reuniones, en las que se discuten los aspectos del proceso que contribuyen al desarrollo y se descarta lo que no genere valor. El estado del proyecto, determina su avance en base a lo que se ha planificado y es un indicador que ayuda a detectar posibles diferencias que pueden surgir y que cambiarían el rumbo a que apuntaba el proyecto. En la figura 6 se puede ver el detalle interno de cada fase. La flecha que trasciende las tres fases en sentido inverso es el ciclo del aprendizaje, lo cual hace ver un cambio en el esquema tradicional en donde existe un ciclo de control para detectar diferencias y corregirlas, es decir que se hace una corrección sobre lo ya trabajado,

mientras que en ASD el cambio es parte del proceso y no es considerado como un error.

Con ASD se considera que la participación de los involucrados en el proyecto es necesaria para aprender, nos da la posibilidad de entender más del negocio y desarrollar una aplicación que mejor satisfaga las necesidades del cliente. (Highsmith, 2000).

4.3. Metodologías Crystal

Las metodologías Crystal son una serie que creó Alistair Cockburn; presentan un enfoque ágil, con énfasis en la comunicación, y aceptan la flexibilidad, lo que las hace ideales para cuando no es aplicable la disciplina requerida por otras metodologías como la de programación extrema.

Crystal “Clear” es la metodología más ágil de la serie y de la que existe más documentación. Da énfasis a la comunicación y entregables livianos. Maneja iteraciones cortas que los clientes pueden revisar, y se disminuye de esta forma la necesidad de productos intermedios. Un usuario forma parte del equipo y se encarga de realizar validaciones en la aplicación y participa en la definición de los requerimientos funcionales y no funcionales del software.

Las personas involucradas eligen aquellos principios que les resultan efectivos y agregan o quitan principios en base al consenso grupal del equipo de desarrollo. Se utiliza una nomenclatura de colores de acuerdo al número de integrantes del equipo.

Los proyectos grandes, que necesitan más coordinación y comunicación, se asocian con colores más oscuros. También se puede realizar una clasificación con colores más oscuros de acuerdo a los proyectos en los que un fallo pueda causar mayores problemas. (Cockburn, 2004).

- Clear: equipos de 9 personas o menos.

- Amarillo: 10 a 20 personas.
- Naranja: 21 a 50 personas.
- Roja: 51 a 100 personas.
- Lila: 101 a 200 personas.
- Celeste: 201 a 500 personas.
- Azul: 501 o más personas. (Cockburn, 2004).

4.4. Propiedades de las metodologías Crystal

4.4.1. Entregas frecuentes, en base a un ciclo de vida iterativo e incremental. De acuerdo al proyecto, puede haber desde entregas semanales y trimestrales. El tiempo no se limita estrictamente a un número de semanas específico. Se entrega una parte del software ya probado por los usuarios.

4.4.2. Mejora reflexiva: es la mejora continua que debe existir en el proyecto; las iteraciones ayudan a ajustarlo y mejorarlo.

4.4.3. Comunicación cara a cara: esto se refiere a que el equipo debe estar en una misma ubicación física, para que haya una comunicación entre una y otra persona. Esto, permite mayor fluidez entre las ideas y colaboración para el proyecto.

4.4.4. Seguridad personal: cualquiera puede expresar su opinión, la cual es valiosa y aporta mucho para el desarrollo del proyecto. La equivocación de alguien puede ser cubierta por los demás, como ayuda de todo el equipo hacia una persona.

4.4.5. Enfoque: se recomienda que durante períodos de 2 horas no se interrumpa al equipo para avanzar con el proyecto y que existan objetivos y prioridades bien definidos, para asignar tareas concretas.

4.4.6. Fácil acceso a usuarios expertos: las metodologías Crystal no exigen que los usuarios estén continuamente junto al equipo de proyecto, ya que no todos los clientes disponen del tiempo para hacerlo, aunque si se recomienda que haya reuniones semanales para que den sus puntos de vista respecto a la aplicación.

4.4.7. Entorno técnico con pruebas automatizadas: gestión de la configuración e integración continua. Las pruebas automáticas dan mayor confiabilidad al software. (Cockburn, 2004)

4.5. Desarrollo impulsado por características

Esta metodología más conocida por su nombre en inglés como “Feature Driven Development” y sus siglas FDD, fue creada por Jeff De Luca y Peter Coad en 1998, FDD se estructura alrededor de la definición de features o características que representan la funcionalidad del sistema. Las características tienen un alcance corto, lo que permite su implementación en un par de semanas. Hay una jerarquía de características, y se agrupan por aspectos comunes del negocio.

Una de las ventajas de centrarse en las características del software es el poder formar un vocabulario común que ayude a los desarrolladores a tener una comunicación fluida con los clientes y converger en una visión común del negocio para aplicarlo en el desarrollo. Se preocupa por la calidad, por lo que incluye un monitoreo constante del proyecto, contrarresta situaciones como el exceso en el presupuesto, fallas en el programa o el hecho de entregar menos de lo deseado. Se enfoca en las fases de diseño y no tanto en los requerimientos. (Palmer & Felsing, 2002).

4.6. Actividades de la metodología FDD

4.6.1. Desarrollar un modelo global: la implementación de la arquitectura del software se debe crear paralelamente junto con el desarrollo del

software. Propone un conocimiento global de la aplicación, el entendimiento del negocio, un primer bosquejo de las características del software, y la definición de restricciones y requerimientos no funcionales. Se crea: un documento similar al de “visión” en donde se plasman los objetivos del proyecto, un documento con los requerimientos no funcionales detectados, y, por último, un documento de arquitectura en donde está el diseño del sistema.

4.6.2. Construir una lista de características: se agrupan jerárquicamente las características para estructurar el desarrollo; las prioridades son: A (debe tener), B (sería útil tener), C (agregar si es posible), D (futuro). Se pondera la importancia de cada una para su posterior implementación. (Palmer & Felsing, 2002).

4.6.3. Planificar por característica: se utiliza la lista de prioridades y se establecen tiempos para las futuras iteraciones. En esta actividad participan el líder de proyecto y el líder de desarrollo. A medida que se realiza la planificación se establecen fechas de finalización de cada iteración. Se delega responsabilidad a los programadores y cada quien es dueño de una característica del sistema.

4.6.4. Diseñar por característica: están relacionadas con la parte productiva del proceso en que se construye la aplicación de manera incremental. Se inicia con el diseño que usa las características correspondientes a la iteración. El equipo de desarrollo identifica las clases, atributos y métodos que realizan la funcionalidad requerida. Con los diagramas de secuencia de UML, se verifica que el diseño pueda ser implementado.

4.6.5. Construir por característica: se desarrollan las clases definidas en la actividad anterior. Cada programador implementará los métodos de

las clases que tiene a su cargo, se extienden las clases base de prueba para construir las pruebas unitarias. Una vez que la clase pasa todas las pruebas, se inspecciona el código.

Es recomendable una reunión semanal entre el líder del proyecto y el de desarrollo; esta, debe ser una reunión breve, de no más de 30 minutos, en la cual se reporta el avance de cada característica de grupo o persona a cargo. (Palmer & Felsing, 2002).

4.7. Desarrollo de software ajustado

Esta metodología, más conocida por su nombre en inglés como “Lean Software Development” y sus siglas LSD, tiene su origen en Toyota, es una estrategia de fabricación aplicada con mucho éxito en Japón y ahora famosa en el mundo del software. En la década de 1950, la industria japonesa se recuperaba de la segunda guerra mundial; logró aplicar a las fábricas de carros los conceptos de calidad en la producción.

Taiichi Ohno (1912 – 1990) fue quién inicio la aplicación de este método en Toyota, cuya estrategia se fundamentó en tres bases:

- Construir sólo lo necesario.
- Eliminar todo aquello que no añade valor.
- Parar si algo no va bien, principio de cero defectos.

Mary y Tom Poppendieck crearon esta metodología aplicada al desarrollo del software. La metodología tiene como objetivo eliminar desperdicios y seleccionan aquellas características que realmente aportan valor; da especial importancia a la velocidad y la eficiencia. Se puede aplicar a todos los ámbitos, desde el propio desarrollo hasta en la misma empresa requirente, se incluye a clientes y proveedores. Esta metodología considera los riesgos del proyecto, que manejados

oportunamente se convierten en mejoras al cliente. Es utilizada en numerosos proyectos de telecomunicaciones de Europa. (Poppendieck & Poppendieck, 2003).

4.8. Principios del desarrollo de software ajustado

4.8.1. Eliminar el desperdicio: quitar del proceso y el producto todo aquello que no aporta valor al cliente.

4.8.2. Calidad integrada: El desarrollo debe realizarse desde el inicio con calidad. Las acciones correctivas deben realizarse conforme se detectan, debe haber un enfoque preventivo, que evite la posibilidad de errores.

4.8.3. Crear conocimiento: el desarrollo de software es un proceso de creación de conocimiento que evoluciona conforme su desarrollo. Es importante la contribución del conocimiento de todos los integrantes del proyecto para dar respuesta a los cambios de forma rápida y con calidad. Compartir el conocimiento con otros integrantes del equipo hace mejorar el desempeño en general.

4.8.4. Aplazar las decisiones: media vez hay incertidumbre en los requerimientos o en alguna etapa del proceso, es aconsejable retrasar las decisiones y es preferible recopilar la mayor cantidad de información posible, para poder actuar con mayor certeza y con información clara.

4.8.5. Entregar tan rápido como sea posible: es necesario hacer el desarrollo del producto tan pronto como se tenga una idea clara de lo que se requiere.

4.8.6. Respetar a las personas: los equipos de trabajo deben contribuir al respeto, para fomentar un ambiente agradable de trabajo. Establecer

metas razonables, que puedan alcanzarse y permitan a las personas auto-organizarse para conseguirlas.

4.8.7. Optimizar el conjunto (visión global): se debe tener una visión de conjunto que ayude en la integración global de todos los componentes que contribuyen al proyecto. (Poppendieck & Poppendieck, 2003)

4.9. Método de desarrollo dinámico de sistemas

Esta metodología, más conocida por su nombre en inglés como “Dynamic Systems Development Method” y sus siglas DSDM, remonta sus inicios a 1994, cuando un consorcio de 17 personas se reunió en Inglaterra con el objetivo crear una metodología que pudiera ser utilizada en proyectos del tipo de desarrollo rápido de aplicaciones (DRA). Al usar las mejores prácticas que se conocían en la industria y la experiencia de sus fundadores, hizo que la primera versión de DSDM saliera a principios de 1995, la cual tuvo muy buena aceptación en la industria del software. (Stapleton, 1997).

4.10. Principios de la metodología DSDM

- El involucramiento del usuario es imperativo.
- Los equipos tienen la potestad de tomar decisiones.
- Se enfoca en la entrega frecuente de productos.
- Es esencial cumplir con las expectativas del negocio, para la aceptación de los entregables.
- El desarrollo iterativo e incremental es necesario para la entrega del software.
- Todos los cambios durante el desarrollo son reversibles.
- Los requerimientos están especificados a un alto nivel.
- Las pruebas se integran a través del ciclo de vida.

- Es esencial un enfoque colaborativo y cooperativo entre todo el equipo.

4.11.Fases de la metodología DSDM

Estudio de factibilidad: Se determina si la metodología se ajusta al proyecto que se realizará. Durante el estudio del negocio se involucra al cliente, para entender de buena forma la operatoria que el sistema debe automatizar. Este estudio brinda la base necesaria para iniciar con el desarrollo y define las características generales que deberá contener el software. Posteriormente, se inician las iteraciones durante las cuales se detallan las características identificadas anteriormente y se realiza su diseño.

Lo que diferencia a este método son los principios alrededor de los que se estructura, los cuáles hacen énfasis en los equipos de desarrollo, en las reuniones con el cliente y en las entregas frecuentes de productos. (Stapleton, 1997).
Es conveniente utilizar esta metodología en los siguientes casos:

- Funcionalidad interactiva que utiliza la interface del usuario.
- Usuarios finales comprometidos con el proyecto.
- Complejidad media de la aplicación.
- La aplicación puede dividirse en componentes funcionales más pequeños.
- Proyecto con limitantes de tiempo.
- Requerimientos flexibles y especificados de forma muy general.

El método propone realizar un conjunto mínimo de modelos necesarios para entrega satisfactoria del software y facilidad en el mantenimiento. Estos modelos se definen antes del inicio del desarrollo, y se revalidan con cada iteración.

Cada fase se subdivide en “investigar, refinar y consolidar”. Durante la investigación se chequea que las actividades coincidan con la arquitectura del sistema. Se fijan los objetivos de la iteración y los entregables.

4.11.1. La etapa de refinar consiste en el desarrollo de lo planificado de acuerdo a las prioridades especificadas; posponen los de menor prioridad para las ultimas iteraciones.

4.11.2. La fase, de consolidar consiste en completar los entregables y verifican la calidad de los mismos. Se valida que se cumplió con los requerimientos definidos durante la fase de investigar.

Con esta metodología se identifican roles como el de “visionario” quien es el encargado de asegurar que se satisfacen las necesidades del negocio; “usuario embajador “que es el cliente, quien brinda el conocimiento del negocio y define los requerimientos del software; “coordinador técnico” que es la persona encargada de la arquitectura y validación de estándares técnicos.

Se hacen reuniones para evaluar los requerimientos del software y realizar prototipos en los que se pueden detectar ambigüedades, también ayuda para que exista una buena comunicación entre los analistas y usuarios. El enfoque propuesto consiste en la utilización de un prototipo evolutivo, el cual se refina hasta tener la aplicación deseada. Los prototipos cumplen con las etapas de: “negocio, usabilidad, performance, capacidad, y diseño”. (Stapleton, 1997).

5. Metodologías ágiles mejor reconocidas

5.1. Programación Extrema

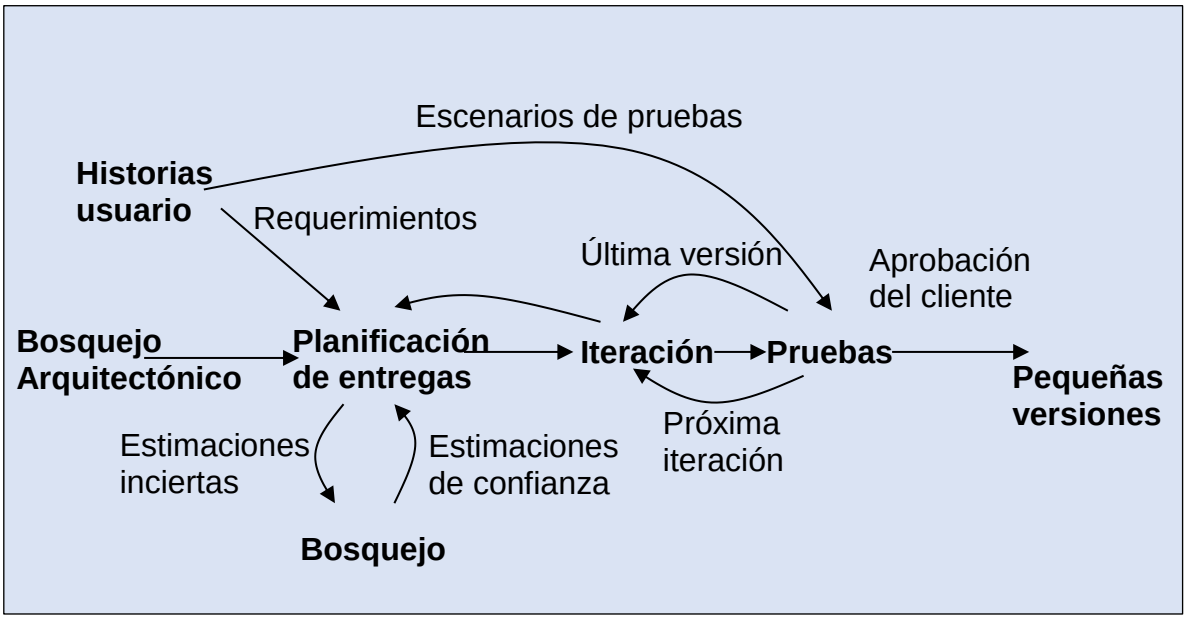
Esta metodología más conocida por su nombre en inglés como “Extreme Programming” y sus siglas XP. Fue creada por Kent Beck en 1990 y Ward Caunnigham. Su proceso se describe en la figura 7.

“Dejar que el desarrollo de software sea divertido, simple, flexible, predecible, con pocos riesgos, eficiente y más científico”. (Beck, 2000).

La programación extrema es una disciplina de desarrollo de software enfocada hacia la satisfacción de las necesidades del cliente y permite cambiar en cualquier momento los requerimientos; también, fomenta el trabajo en equipo de todos sus integrantes, como lo son los administradores del proyecto, los desarrolladores y los clientes. Una de las características importantes de esta metodología es el fundamento de las pruebas del desarrollo, lo que implica que cada programador escribe sus pruebas junto al código, agregándolas en el proceso de integración continua.

La programación extrema es un proceso evolutivo que utiliza iteraciones cortas lo que permite al desarrollador enfocarse en la iteración actual sin realizar nada con anticipación. Así, se obtiene, una metodología disciplinada y adaptable. (Beck, 2000).

Figura 7. Proyecto típico de programación extrema.



Fuente: Wells, 2009.

5.2. Valores de la programación extrema

5.2.1. Simplicidad: “Los programadores extremos hacen una cosa de la manera más simple, de tal forma que esta pueda funcionar” (Anderson, Jeffries, & Hendrickson, 2001). Es mejor tener conceptos simples, porque se entienden fácilmente, a utilizar algo muy complicado y que nadie entienda.

5.2.2. Comunicación: se fomenta la comunicación entre los involucrados en el proyecto, desde clientes hasta jefes de proyecto. Esto se logra por medio de prácticas aplicables en corto tiempo: entre ellas, están las pruebas unitarias, programación en parejas y la estimación de tareas.

5.2.3. Reunión: es la interacción entre los clientes y desarrolladores, esta constituye la confidencia y ayuda a eliminar las confusiones y crea conciencia de todas las actividades del proyecto, en las reuniones se trata la corrección de errores.

5.2.4. Coraje: “Tener una acción efectiva ante los problemas” (Beck, 2000). Los desarrolladores necesitan coraje para enfrentar situaciones de la vida real. La simplicidad y comunicación dan valor agregado.

La comunicación da coraje porque se expone lo que se piensa y como se cree que se puede solucionar un problema. Sugiere porque el programador expondrá lo que no le parece y dirá lo que a su criterio es lo mejor. Simplifica, porque cuando hay posibilidad de reducir algo, lo intenta.

5.2.5. Respeto: este enfatiza la necesidad del respeto entre todos los miembros del equipo. Se fomenta el respeto por todas las ideas de cada miembro del equipo; las contribuciones de cada miembro son valoradas y respetadas para dar coraje entre ellos. (Beck, 2000).

5.3. Principios de la programación extrema

5.3.1. Reunión: trabajar en conjunto con el cliente ayuda a entender mejor los requerimientos como, por ejemplo, la creación de una interfaz adecuada a sus necesidades.

Se debe hacer pequeños ciclos de tiempo para establecer si verdaderamente se está cumple con los requerimientos del cliente.

5.3.2. Asumir simplicidad: se debe asumir que cada problema se puede resolver con simplicidad; esto, implica enfocarse en la iteración actual, sin considerar lo que pueda pasar en iteraciones futuras.

5.3.3. Realizar cambios incrementales: los problemas se solucionan con una serie de cambios, se toma en cuenta la planificación, diseño, desarrollo y pruebas para obtener, una solución mejorada.

5.3.4. Adopción del cambio: el equipo debe estar dispuesto siempre a esperar cambios y estar listos para adoptarlos. Se debe tener una estrategia que permita establecer alternativas de acuerdo al cambio que se necesite realizar.

5.3.5. Hacer trabajo de calidad: la calidad tanto en la codificación como en el sistema en general, tiene como consecuencia una gran satisfacción en el cliente, lo cual se logra, en parte, por medio de las pruebas antes del desarrollo. (Beck, 2000).

5.4. Prácticas de la programación extrema

Se debe practicar los valores; por ejemplo, al programar en parejas tiene que practicarse la buena comunicación y la simplicidad en el sistema. A continuación, las prácticas de la metodología XP:

5.4.1. El juego de la planeación: determina el alcance de la iteración actual, y las prioridades del cliente para poder implementarlas. Se establece

la duración de cada tarea, la organización de la cultura de trabajo y la planificación de cada iteración.

5.4.2. Versiones pequeñas: esto ayuda a poner en marcha el sistema en poco tiempo, que va desde un día a un mes; sin embargo, hay situaciones donde se puede requerir de más tiempo, de acuerdo a la complejidad del proyecto, aunque el máximo podría ser un referente de 3 a 4 meses.

5.4.3. Diseño simple: el diseño simple ayuda en la entrega rápida de una versión funcional del producto a diferencia de si se realiza algo muy complejo que sería muy difícil de entregar a tiempo, quizás no sea lo que el cliente quería y luego lo cambie.

5.4.4. Pruebas: las pruebas son imprescindibles en la programación extrema, ya que con ellas se garantiza el éxito del proyecto, por lo regular a nivel de codificación se crean las pruebas antes del código. Se hace la prueba, se asume que existe lo que se desea obtener y luego se codifica para alcanzarlo.

5.4.5. Integración continua: la integración del proyecto se realiza en lapsos cortos de tiempo; normalmente, se hace a diario, lo que ayuda a eliminar rápidamente los problemas encontrados. No se tiene que invertir mucho tiempo a la hora de integrar ya que si se encuentran muchos fallos en el código cuando se integra y no se logra resolver rápidamente, se debe descartar. (Beck, 2000).

5.4.6. Refactorización: “Si el código apesta, cámbialo” (Beck, 2000). Si el código es difícil de entender o modificar, se debe reemplazar por otro mejor. Es mejor reescribir el código, lo que se llama “refactorizar”. La refactorización continua ayuda a crear cambios más rápidamente.

5.4.7. Metáfora: esta define como conceptualiza el equipo al sistema, escribiéndolo en un lenguaje relevante para el negocio.

5.4.8. Programación en parejas: el código se desarrolla en parejas, en la misma computadora; esto, contribuye a que ambas personas cooperen mutuamente para alcanzar la mejor solución al problema. Por lo regular se coloca a un programador con mucha experiencia y a otro con menos experiencia.

5.4.9. Propiedad colectiva del código: es importante tener una estandarización dentro del código para que cualquier integrante del equipo pueda modificar fácilmente algún cambio requerido.

5.4.10. Cuarenta horas semanales: es el tiempo adecuado que las personas deben trabajar en el proyecto, ya que trabajar de más puede reducir el desempeño del equipo y, en tal situación, difícilmente se alcanzaría un proyecto con buena calidad. Una buena planificación del tiempo ayuda mucho.

5.4.11. Cliente en el sitio: el cliente debe estar junto al equipo de desarrollo, para responder cualquier duda que tengan los desarrolladores y establecer las prioridades de acuerdo a sus requerimientos.

5.4.12. Estándares de codificación: cualquiera del equipo debe entender el código escrito por cualquier desarrollador; a esto, se le conoce como colectividad del código, lo que reduce el tiempo al momento de requerir alguna modificación. (Keyes, 2015).

5.5. Scrum

Scrum define un proceso empírico, iterativo e incremental de desarrollo que aprovecha la naturaleza caótica del desarrollo de software y la utilización de

prácticas para el manejo de la impredictibilidad y el riesgo. Sus orígenes se remontan a 1993 cuando el Dr. Sutherland había sido contratado por la compañía Easel como vicepresidente de tecnología de objetos.

Los ejecutivos de la empresa requerían el desarrollo de una nueva línea de productos de software en 6 meses, pero por lo limitante del tiempo era evidente que si se utilizaba la tradicional metodología cascada no lograrían hacerlo. Fue entonces cuando uno de sus desarrolladores llevó al equipo un artículo publicado en 1986 en la revista de negocios de Harvard titulado “El juego del nuevo desarrollo de productos” escrito por dos profesores de administración japoneses, Hirotaka Takeuchi e Ikujiro Nonaka. (Sutherland, 2016).

Este artículo describía las mejores prácticas utilizadas en las compañías más productivas e innovadoras de aquel tiempo: Honda, Fuji-Xerox, 3M, Hewlett-Packard, entre otras. Afirmaban que la manera de desarrollar productos como el del sistema de planeación gradual de programas de la NASA, si era un sistema en cascada, presentaba fallas en su origen.

En cambio, las mejores compañías seguían un proceso de desarrollo más rápido y flexible. Sus equipos tenían autonomía, autoridad para tomar decisiones, su dirección no daba órdenes, más bien los ejecutivos eran líderes de servicio y facilitadores que quitaban los obstáculos. (Sutherland, 2016).

Los profesores japoneses comparaban el trabajo en común con un equipo de rugby y decían que los mejores equipos actúan como en un scrum: “La pelota circula entre los integrantes del equipo a medida que éste avanza por el campo como una unidad” (Takeuchi & Nonaka, 1986).

En 1986, la fecha en que se publicó ese artículo, siete años antes de que el Dr. Sutherland iniciara la creación de esta metodología aplicada al software, causó revuelo y asombró, pero nadie hizo nada al respecto.

Los gerentes, principalmente los estadounidenses, no le encontraban mucho sentido, a pesar de que Toyota aumentaba rápidamente su participación de mercado con la aplicación de este enfoque. A pesar de que el artículo de los profesores japoneses se basaba en manufactura, el Dr. Sutherland consideró apropiado aplicarlo al desarrollo del software de la nueva línea que tenían que realizar y motivado por la idea del trabajo en equipo óptimo, a lo cual estaba muy acostumbrado. (Sutherland, 2016).

Fue así como nació Scrum, se entregó dicha nueva línea de productos de software a tiempo, dentro del presupuesto y con menos errores que todos los productos previos, A partir de ahí, el Dr. Sutherland ha afinado esa metodología para su uso en toda clase de empresas. En 1995 presentó en coautoría con Ken Schwaber, en una conferencia en la Asociación de los Sistemas Informáticos, un artículo titulado “Proceso de desarrollo Scrum”, que codificaba esas prácticas; por ello, también se considera a Schwaber como coautor de esa metodología. (Sutherland, 2016).

Scrum es un método iterativo e incremental que se enfoca en las prácticas y valores de la gestión de proyectos más que en las disciplinas del desarrollo. En el inicio del proyecto se define la lista de requerimientos (Product Backlog), que contiene todos los requerimientos funcionales y no funcionales que deberá satisfacer el sistema. Los mismos son especificados de acuerdo a las convenciones de la organización ya sea mediante: características (features), casos de uso, diagramas de flujo de datos, incidentes, tareas y otros.

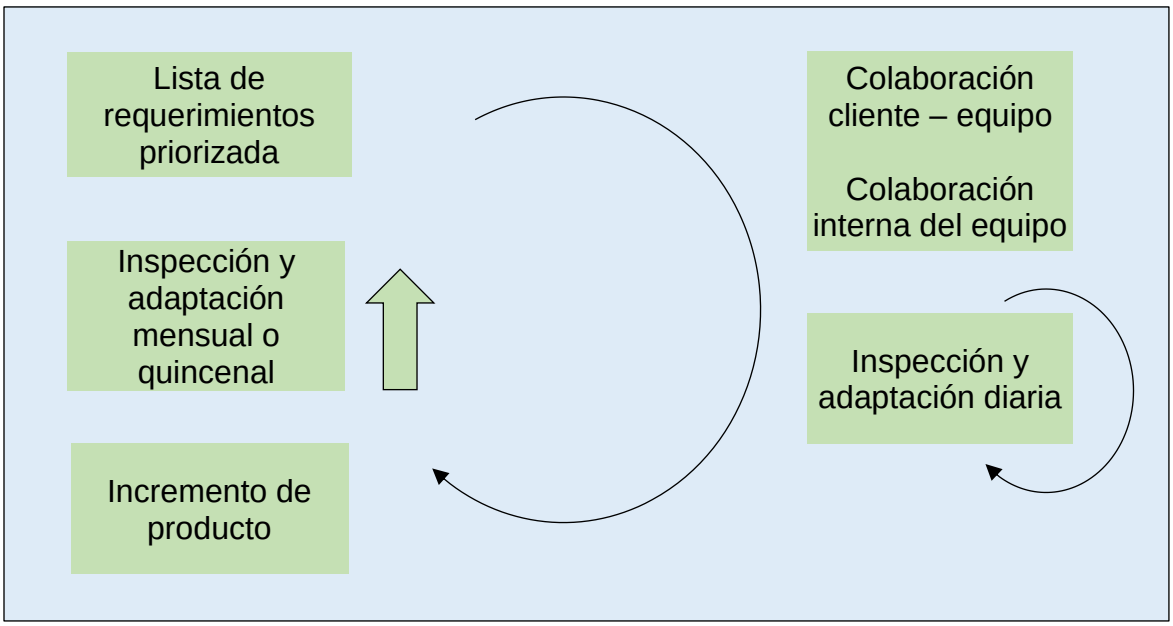
La lista de requerimientos se define mediante reuniones de planeamiento con los involucrados en el proyecto. A partir de ahí se definen las iteraciones, conocidas como sprints en Scrum, en las que evoluciona la aplicación.

Cada iteración tiene su propia lista de requerimientos que es un subconjunto de la lista general con los requerimientos a ser construidos en ese ciclo de trabajo en específico. La duración recomendada de cada fase es de 1 mes. Dentro de cada ciclo de trabajo, el facilitador lleva a cabo la gestión de la iteración y convoca

diariamente a la reunión de Scrum en la que se presenta el avance diario; la duración debe ser menos a 15 minutos, con el propósito de tener ralimentación sobre las tareas de cada integrante del equipo y los obstáculos que afrontan. Al final de cada iteración, se realiza una revisión de la fase, en la que se evalúan los artefactos desarrollados y comenta el planeamiento de la próxima fase.

La metodología define algunos roles y artefactos que contribuyen a tener un proceso que promueve reuniones para mitigar cualquier riesgo que pueda presentarse. (Sutherland, 2016).

Figura 8. Manejo del proyecto con Scrum.



Fuente: Albaladejo, 2013.

La intención de Scrum es corregir problemas y mitigar riesgos tempranamente. Su uso se extiende cada vez más dentro de la comunidad de metodologías ágiles; el mismo se puede combinar con otras metodologías como XP para completar sus carencias. (Schwaber & Beedle, 2001) (Schenone, 2004).

Con la metodología Scrum el cliente se entusiasma y se compromete con el proyecto, da puntos de vista sobre el funcionamiento que desea del software que

se desarrolla, esto le permite ver el crecimiento del software conforme cada iteración.

El cliente puede, en cualquier momento, realinear el software con los objetivos de negocio de su empresa, ya que puede introducir cambios funcionales o de prioridad en el inicio de cada nueva iteración, sin ningún problema. Esta metodología promueve la innovación, motivación y compromiso del equipo que forma parte del proyecto, por lo que los profesionales encuentran un entorno propicio para desarrollar sus capacidades.

5.6. Beneficios de Scrum

5.6.1. Cumplimento de expectativas: el cliente establece sus expectativas e indica el valor que le aporta cada requerimiento, el equipo técnico los estima y con esta información el dueño del producto establece su prioridad. En las demostraciones de la fase el dueño del producto comprueba que efectivamente los requerimientos se han cumplido y se brindan sugerencias correspondientes al equipo.

5.6.2. Flexibilidad a cambios: alta capacidad de reacción ante los cambios de requerimientos generados por necesidades del cliente o evoluciones del mercado. La metodología está diseñada para adaptarse a los cambios de requerimientos que conllevan los proyectos complejos.

5.6.3. Resultados anticipados: el cliente puede empezar a utilizar las funcionalidades más importantes del proyecto antes de que esté finalizado el software por completo.

5.6.4. Mayor calidad del software: la metodología de trabajo y la necesidad de obtener una versión funcional después de cada iteración, ayuda a tener un software de calidad superior.

5.6.5. Mayor productividad: se elimina la burocracia y se motiva al equipo, se da autonomía a las personas para auto-organizarse.

5.6.6. Maximiza el retorno de la inversión: producción de software únicamente con las prestaciones que aportan mayor valor al negocio. (Schwaber & Beedle, 2001).

5.6.7. Predicciones de tiempos: mediante esta metodología se conoce la velocidad media del equipo por iteración, conocidos como puntos historia, con lo que es posible estimar fácilmente para cuando se dispondrá de una determinada funcionalidad que todavía está en la lista de requerimientos.

5.6.8. Reducción de riesgos: se reducen riesgos al conocer el tiempo medio que tarda el equipo de desarrollo y las entregas funcionales de mayor valor al inicio del proyecto. (Schwaber & Beedle, 2001).

III. JUSTIFICACIÓN

En la estimación de un proyecto de software se debe considerar diversos factores como: el tamaño del proyecto, cantidad de personas que realizarán el proyecto, recursos disponibles, entre otros. El no realizar una buena estimación del proyecto hace que el proyecto no pueda entregarse en el tiempo estipulado. Más aún, cuando se pretende realizar completamente el proyecto sin tener entrega parcial del software final, el cliente no sabe si en realidad el sistema cumplirá con lo solicitado.

Durante el transcurso del desarrollo del sistema no existe comunicación adecuada entre el cliente y los demás involucrados en el proyecto, lo que provoca que se tome un rumbo equivocado y que el sistema no funcione de acuerdo a las expectativas del cliente.

Por las situaciones planteadas anteriormente, se considera conveniente el uso de la metodología Scrum, que permite administrar efectivamente el proyecto por medio de la flexibilidad y entregas parciales completamente funcionales del sistema, con lo cual el cliente puede observar si cumple con sus expectativas. Con esto se obtiene una mejor estimación debido a que el proyecto se divide en fases de 2 a 4 semanas, de acuerdo a la complejidad y tamaño del proyecto.

Con Scrum existe una comunicación constante entre el cliente y los demás involucrados en la realización del proyecto; esto, permite que el cliente, en cualquier momento, pueda realinear el sistema con los objetivos del negocio y empresa y dar sus prioridades.

Ese alto grado de comunicación promueve la innovación, motivación y compromiso del equipo involucrado en el proyecto, lo cual conlleva a obtener un sistema de calidad que satisfaga las expectativas del cliente.

IV. OBJETIVOS

1. Objetivo general

1.1. Elaborar un estudio sobre la administración efectiva de proyectos de software por medio de la metodología Scrum, en busca de que las empresas de sistemas optimicen el tiempo y cumplan con la entrega planificada de sus proyectos.

2. Objetivos específicos

2.1. Plantear el uso de la metodología Scrum para gestionar efectivamente un proyecto de software, a través del trabajo en equipo, buena comunicación con los clientes y flexibilidad en la administración.

2.2. Proponer la realización del sistema final por fases, entregadas conforme a prioridades y especificaciones establecidas por el cliente, para contribuir a una buena estimación del proyecto de software y cumplir con la entrega a tiempo.

V. METODOLOGÍA

1. Tipo de estudio

1.1. Investigación bibliográfica

En este documento se utilizó la investigación bibliográfica, principalmente con los libros que describen los detalles de la metodología SCRUM para el desarrollo de sistemas; además, se usaron tesis, artículos y videos publicados en internet que sustentan un estudio académico formal, los cuales ayudaron en el enriquecimiento del contenido del presente documento.

2. Método de recolección de datos

2.1. Revisión bibliográfica:

2.1.1. Libros: Los libros en donde se abordaba principalmente la metodología Scrum para el desarrollo de sistemas fueron de gran utilidad para elaborar un análisis detallado y describir cómo utilizar dicha metodología en el presente documento de investigación.

2.1.2. Tesis: Las tesis se utilizaron principalmente para la elaboración del marco teórico, fueron de gran utilidad para comprender de mejor forma los temas relacionados a las metodologías de desarrollo de sistemas.

2.1.3. Artículos: Los artículos publicados en internet ayudaron a tener una visión amplia de cómo se utiliza la metodología Srum y en base a ello se eligieron los aspectos más sobresalientes que ayudan en la investigación sobre esta metodología.

VI. RESULTADOS

Administración efectiva de Proyectos de software con Scrum

Pablo César Paniagua González

Guatemala, octubre de 2017

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El desarrollo mundial del software ha utilizado un esquema tradicional en donde se lleva a cabo una rigurosa definición de actividades como el modelado, planificación y documentación detallada, que ha sido inefectivo en grandes proyectos; como el proyecto Sentinel del FBI.

Los proyectos de software son cambiantes, y su desarrollo exige tiempos reducidos y una alta calidad, para lo cual es útil la metodología ágil Scrum. Su nombre proviene del juego de rugby y se refiere a la forma en que el equipo se desempeña en común para mover la pelota en la cancha, con acoplamiento, en busca del mismo propósito y metas claras.

Por medio de la metodología Scrum se busca que el proyecto de software se entregue a tiempo, lo cual se logra con la colaboración de trabajo en equipo, buena comunicación con el cliente, flexibilidad en la planificación, entregas parciales del software final y resolución inmediata de problemas que surgen conforme el avance del proyecto.

En síntesis, se realizó un estudio sobre la agilización del proceso de desarrollo de software por medio de la metodología Scrum; en él se usó como apoyo, diferentes fuentes bibliográficas como artículos, tesis, videos, sitios de internet y principalmente libros de ese campo técnico, en busca de orientar a las empresas de sistemas para que la puedan implementar.

OBJETIVOS

1. Objetivo general

1.1. Elaborar un estudio sobre la administración efectiva de proyectos de software por medio de la metodología Scrum, para que las empresas de sistemas conozcan sobre el uso de esa metodología.

2. Objetivos específicos

2.1. Dar a conocer el uso de la metodología Scrum por medio de la definición de las actividades y características que conlleva esta metodología, que permite gestionar efectivamente un proyecto de software.

2.2. Proponer la subdivisión en módulos del sistema final, con entregas conforme a prioridades y especificaciones establecidas por el cliente, que permitan mejorar la estimación de un proyecto de software.

RESULTADO 1

1. Estudio sobre la administración efectiva de proyectos de software por medio de la metodología Scrum

Scrum es una metodología en donde se hace el desarrollo del software iterativo e incremental; este se divide en ciclos de trabajo llamados sprints, con iteraciones de 1 a 4 semanas consecutivas, al iniciar el ciclo de trabajo. Un equipo se encarga de seleccionar los requerimientos del cliente, de una lista priorizada. Todos los días, el equipo se reúne brevemente; es aconsejable una reunión de no más de 15 minutos, para informar del progreso, y actualizar el avance del proyecto.

Al final de la fase, el equipo revisa y muestra al cliente el software realizado y probado durante esta fase, el cliente da sus comentarios y observaciones para incorporarlos en la siguiente fase, hasta que se finaliza con el software completamente. En la siguiente gráfica se ven los roles y eventos principales de Scrum. En la figura 1 se describen los roles y eventos principales de Scrum.

2. Roles de Scrum

En Scrum hay 3 roles principales

- El dueño del producto
- El equipo
- El encargado del equipo

2.1. El dueño de producto (DP)

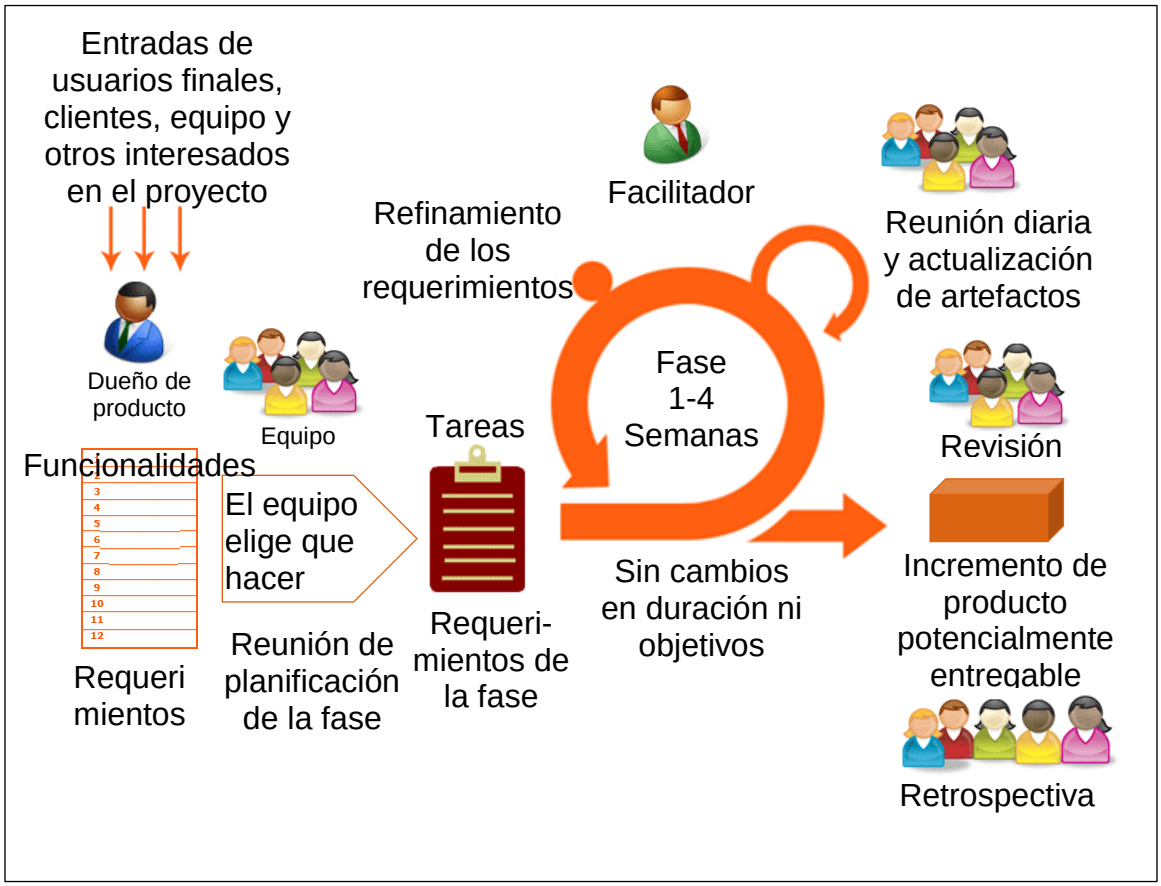
Se encarga de maximizar el retorno de la inversión por medio de identificar y priorizar las funcionalidades del producto. En la lista priorizada se definen las

funcionalidades que van al inicio en cada fase y, continuamente, se ajusta la lista de acuerdo a las necesidades del negocio.

El dueño del producto tiene la responsabilidad de las pérdidas y ganancias del producto, al elegir en cada fase los elementos que generen más valor para el negocio con el menor costo. El dueño del producto es un representante del cliente.

El rol del dueño del producto es parecido al rol de jefe del producto; sin embargo, el dueño del producto es diferente al tradicional jefe de producto porque interactúa frecuentemente con el equipo, establece personalmente las prioridades y revisa el resultado en cada iteración que va de 1 a 4 semanas. En Scrum solo una persona puede tener la autoridad de “dueño del producto”. (Rubin, 2013).

Figura 1. Roles y eventos principales de Scrum.



Fuente: Scrum Organization, 2016.

2.2. El equipo

Se encarga de realizar el software requerido por el cliente, en tanto equipo multi-funcional que tiene la capacidad de entregar el software planificado para cada fase. Es un equipo autónomo, en el que se gestionan las tareas correspondientes dentro del mismo sin estar necesariamente dirigidos por un jefe de equipo, sino que, más bien, todos los integrantes se auto gestionan. El equipo decide a que actividades comprometerse con tal de cumplir con todo lo que se propuso.

El equipo en Scrum incluye analistas, desarrolladores, diseñadores de interface y encargados de pruebas. El equipo desarrolla el producto y da sugerencias al dueño del producto sobre cómo quedaría de mejor forma el software. Se asume que el equipo está dedicado 100% al trabajo asignado durante la fase; evita hacer varias tareas en diferentes proyectos y también evita cambiar miembros del equipo.

Los grupos de desarrollo grandes se deben organizar en varios equipos, en los que cada uno realice diferentes funcionalidades del software, se trabaja en forma coordinada y, de esta forma, optimiza los recursos para obtener el software en un tiempo óptimo. (Rubin, 2013).

2.3. El encargado del equipo

La función de este rol no es la de tomar decisiones y gobernar al equipo, es decir no es el jefe del equipo o jefe de proyecto, no le dice a la gente las tareas que tienen asignada, sino que facilita el proceso, ayuda al equipo que es auto gestionable; es más bien, un rol que vela por la integridad de la metodología y su tarea principal es ver que las actividades de Scrum se realicen de forma correcta. Ello incluye al equipo, dueño de producto y gerencia; que haya un buen ambiente en el equipo y que no existan intervenciones externas al mismo. Es el portavoz del equipo que comunica lo que sea necesario hacia el exterior y viceversa. El facilitador adapta la

metodología a las condiciones del equipo actual y controla que todo funcione correctamente.

Es recomendable que los equipos tengan un facilitador de tiempo completo, aunque en un equipo más pequeño podría ser un miembro del equipo, a quien debe asignársele una carga de trabajo más ligera. Su experiencia puede haber sido en las áreas de ingeniería, diseño, pruebas, gestión de productos, gestión de proyectos o gestión de calidad.

El facilitador no puede ser la misma persona que el dueño del producto; incluso, algunas veces, es necesario que el facilitador haga ver al dueño de producto que no es posible agregar nuevas funcionalidades a mitad de un ciclo de trabajo.

Si fuera el caso de migrar de una metodología tradicional a Scrum, es conveniente que la persona que era jefe de algún equipo en específico sea asignada a un grupo diferente como facilitador; y esta persona, debe entender la diferencia que existe con Scrum, cambian su forma de pensar y de comunicarse con el equipo ya no como jefe sino como facilitador y gestionar todas las actividades del proyecto.

La Scrum Alliance se dedica a certificar “Scrum Master” si fuera el caso en que la empresa decidiera invertir en certificar a sus colaboradores encargados de realizar el rol de facilitador en sus equipos.

El Scum master ayuda a quitar obstáculos, a resolver problemas por medio de la discusión entre todos los involucrados en la solución, da ideas creativas y guía el desarrollo de habilidades de los integrantes del equipo. (Rubin, 2013).

3. Proceso de Scrum

Inicialmente, el “dueño del producto” da la visión del software a realizar, que luego evolucionará a una lista priorizada de funcionalidades llamada pila de producto. Ver

ejemplo en tabla 1. Este documento de requerimientos es el plan de trabajo del proyecto que cambia conforme avanza el proyecto. Son los requerimientos que desarrollará el equipo en orden de prioridad. Solo hay una lista de requerimientos en donde el dueño de producto tiene que decidir sobre la priorización de todo el sistema.

En el documento de requerimientos se incluyen las funcionalidades nuevas del cliente o las mejoras en caso de ser un sistema ya existente y, si hay fallos en el sistema actual, se incorporan los requerimientos de cambio correspondientes. Los requerimientos se definen en historias de usuario, en donde se describe clara y concisamente la funcionalidad que el usuario final necesita.

El subconjunto de la lista de requerimientos correspondiente al ciclo de trabajo actual se conoce como la “pila de entrega”, que es el objetivo principal del dueño del producto. La lista de requerimientos se actualiza continuamente por el “dueño del producto” quién incorpora cambios en los requerimientos, aporta nuevas ideas y movimientos de los competidores, estima y evita dificultades técnicas, entre otros.

El equipo da al “dueño del producto” las estimaciones del esfuerzo requerido para cada requerimiento de la lista. El mismo es responsable de asignar la estimación que se espera del lado del negocio, sobre cada requerimiento.

El dueño de producto prioriza los requerimientos de acuerdo al esfuerzo que lleva al equipo realizar el proyecto y el tiempo que el cliente estima, según su conveniencia; incluye también estimaciones adicionales de riesgo. Con el tiempo, un equipo puede calcular un promedio de cuantos puntos se implementan por fase y, de esta forma, proyectar una fecha de entrega del software terminado.

Los elementos de la pila de producto pueden variar significativamente en tamaño y esfuerzo, los elementos grandes se dividen durante la definición de la pila de producto o en la reunión de planificación de la fase.

El nivel de detalle en la especificación de cada elemento de la pila depende del dueño de producto y el equipo; lo importante, es no extenderse demasiado, sino que colocar lo que es importante, debe ser puntual en cada detalle.

Tabla 1. Ejemplo de una lista de requerimientos.

				Estimación de esfuerzo del trabajo restante de la fase					
Elemento	**Prioridad**	**Estima-ción del valor**	**Estima-ción de es-fuerzo inicial**	**1**	**2**	**3**	**4**	**5**	**6**
Como comprador, quiero poner un libro en el carrito de la compra.	1	7	5						
Como comprador, quiero quitar un libro del carrito de la compra.	2	6	2						
Mejorar el rendimiento del procesador de transacciones.	3	6	13						
Investigar soluciones para acelerar la validación de tarjetas de crédito.	4	6	20						
Actualizar todos los servidores a Apache.	5	5	13						
Diagnosticar y arreglar los errores de los scripts de procesamiento de órdenes.	6	2	3						
Como comprador, quiero crear y guardar una lista de Regalo.	7	7	40						
Como comprador, quiero añadir o borrar elementos en mi lista de compras.	8	4	20						

Fuente: Deemer, Benefield, Larman, & Vodde, 2009.

Los elementos de alta prioridad que serán implementados próximamente tienden a tener más detalle; mientras, los elementos de baja prioridad y los que están lejos de ser implementados tienen los requerimientos menos detallados. (Sutherland & K., 2012)

3.1. Creación de la fase o ciclo de trabajo

Inicialmente, se realiza la reunión de planificación de la fase en donde el dueño del producto, equipo, e incluso el encargado del equipo, revisan los elementos de alta prioridad de la lista de requerimientos que el dueño de producto está interesado en implementar para esta fase, de tal forma que el equipo se haga una idea de lo que desea el dueño de producto. El dueño de producto y el equipo también revisan la “definición de hecho” que todos los elementos deben cumplir.

Definición de hecho significa codificado con estándares, revisado, implementado con desarrollo orientado a pruebas y probado 100%, integrado y documentado. El equipo se centra en entender qué quiere el dueño de producto.

En reuniones siguientes se ve la planificación detallada de tareas con descripción de los elementos que el equipo decide hacer. El equipo selecciona los elementos de la lista de requerimientos que se entregarán con la fase, y se comienza con los elementos que tienen más prioridad para el dueño de producto. El equipo decide a cuanto trabajo se compromete en vez de que este sea asignado por el dueño de producto, lo que da mayor certeza en la entrega ya que el equipo lo hace con base en su propio análisis y planificación, en vez de que sea hecho por alguien ajeno al equipo. El equipo también puede seleccionar elementos de menos prioridad en la lista, cuando sea útil y correspondan con los de alta prioridad.

En la planificación el equipo debe realizar la misma y estimar cuánto tiempo tiene directamente cada miembro para trabajo relacionado con la fase, diariamente, sin incluir otras actividades. Usualmente va de 4 a 6 horas diarias.

Después se inicia con el primer elemento de la lista de requerimientos, que es el de más alta prioridad y se divide en tareas individuales, que se guardan en el documento llamado “pila de la fase”. El equipo se mueve secuencialmente hacia abajo de la lista de requerimientos de esta forma hasta que se termine la iteración. Al final de la reunión el equipo habrá producido una lista de tareas con las estimaciones de cada tarea correspondiente; es recomendable que cada tarea sea de 6 horas aproximadamente. Los colaboradores tienen diferentes funciones, es decir que no se clasifican con un puesto en específico. (Maximini, 2015).

Los miembros del equipo ayudan donde está el trabajo y en todo lo posible, proactivamente. Si hay muchas tareas de pruebas, entonces todo el equipo puede ayudar. Esto no significa que todos son generalistas; claro está que hay personas entrenadas especialmente en probar, y programar, pero los otros miembros del equipo trabajan conjuntamente y aprenden nuevas habilidades de los demás. Por lo tanto, durante la generación de tareas y estimación de la planificación de la fase, las personas deben ser asignadas para una sola tarea a la vez y, en caso haya la necesidad de realizar más de una tarea a la vez, se debe considerar tareas relacionadas entre sí.

También, al momento de asignar tareas, se debe considerar la experiencia que tenga un miembro del equipo, o bien que haya realizado alguna tarea similar, ya que esto reduce el tiempo en la ejecución de dicha tarea. Se debe usar una herramienta visual de seguimiento de tareas, en forma de un gran tablero de tareas en la pared, donde las tareas se cambian durante el ciclo de trabajo entre columnas etiquetadas con:

- No empezado
- En progreso
- Completado

Cualquier cambio o adición de nuevos requerimientos al desarrollo debe esperar a la siguiente fase. Si aparece alguna circunstancia externa que haga cambiar las prioridades significativamente, e implica que el equipo perdería el tiempo si continua con el trabajo actual, el dueño de producto o el equipo puede terminar la fase, para iniciar una nueva que contenga los cambios correctos. En este caso, se realiza una nueva reunión de planificación de la fase y se da comienzo a la nueva. Es importante hacer notar que estos casos deben ser la excepción. El equipo debe trabajar con la certeza de que sus tareas no cambiarán, para estar claro en una fecha específica de finalización.

El dueño del producto debe claramente definir la prioridad de sus requerimientos para evitar cualquier cambio durante la fase. Al seguir estas reglas de Scrum, el dueño de producto tiene la confianza de que el equipo terminará un conjunto de trabajo claro y realista que se ha definido previamente; está la flexibilidad en el sentido de que puede hacer cualquier cambio antes de que inicie la siguiente fase. En ese momento, adiciones, supresiones, modificaciones y repriorizaciones son factibles. (Maximini, 2015).

3.2. Historias de usuario

Son las descripciones de las funcionalidades que va a tener el software. Son el resultado de la colaboración entre el cliente y el equipo; ellas evolucionan durante el transcurso del proyecto.

En la figura 2 se describe una historia de usuario. Las historias de usuario tienen tres componentes:

- Tarjeta: es una breve descripción escrita que servirá como recordatorio.
- Conversación: es una conversación que servirá para asegurarse de que se ha entendido bien todo, y concretado el objetivo.
- Confirmación: pruebas funcionales para fijar detalles que sean relevantes e indicar cuál será el límite.

Figura 2. Historia de usuario

Precalificación del cliente
Como cliente quiero que se pueda precalificar a un cuentahabiente para ver si es apto para optar a un crédito bancario. Estimación: 5 Prioridad: 2

Fuente: Elaboración propia.

- Id: Identificador de la historia de usuario.
- Título: debe ser descriptivo para la historia de usuario.
- Descripción: descripción resumida de la historia de usuario.
- Estimación: evaluación del costo de implementación en unidades de desarrollo. estas unidades representarán el tiempo teórico, desarrollo/persona, que se haya estimado al inicio del proyecto.
- Prioridad: en la implementación de la historia de usuario, respecto a las demás historias de usuario.

Para priorizar las tareas están los siguientes indicadores:

- Obligatorio: se debe completar este requerimiento para finalizar el proyecto.
- Debido: se debe completar este proyecto por todos los medios, pero el éxito
- del proyecto no depende de él.
- Probable: se debería completar este requerimiento si su implementación no afecta a la consecución de los objetivos principales del proyecto.
- Deseable: se puede completar este requerimiento si sobra tiempo de desarrollo.
- Dependencias: en este apartado se indican los identificadores de las tareas de las que depende. (Cohn, 2004)

3.3. Definición de hecho

La Definición de hecho es un acuerdo del equipo que contiene las condiciones que deben cumplir los elementos de la lista de requerimientos, que se aceptarán en la fase para considerarlos completados. Incluye los aspectos técnicos, de documentación y de pruebas. Cada equipo crea y mantiene su propia definición de hecho, que suele evolucionar y refinarse conforme el equipo integra, perfecciona y automatiza sus prácticas de desarrollo.

En la reunión de verificación se comprueba si se ha dado cumplimiento con la definición de hecho. Scrum no tiene auditorías, es el equipo quien autogestiona su calidad. Algo importante es garantizar que el trabajo se realiza con una calidad uniforme, independientemente del integrante del equipo que lo haga.

Indica si una historia de usuario o tarea está finalizada o no, aunque hay ocasiones en que una tarea no se ha completado o no se ha podido dividir, por lo que se hace necesario informar sobre el porcentaje de avance sobre dicha tarea. Para ello se puede crear una definición de hecho que permite establecer avances parciales sin dejar de tener claro cuando una tarea debe considerarse como finalizada.

Cuando una historia es codificada implica el 50%, y todavía resta completar la creación de los casos de pruebas, obtener la validación del control de calidad, revisar el código por un ingeniero diferente al que creo el desarrollo, generar la documentación necesaria y finalmente mostrar y obtener el visto bueno del software entregado en el ciclo de trabajo. Se considera terminado cuando el cliente lo ha visto y ha dado su aprobación. (Cohn, 2004)

3.4. Reunión diaria de Scrum

Es una reunión corta de aproximadamente 15 minutos que se realiza todos los días, de preferencia al inicio de la jornada. Todo el equipo asiste a la reunión. Para hacerla

corta, se recomienda que todos estén de pie, cada integrante del equipo informa sobre el progreso y los obstáculos; principalmente, se informa lo siguiente:

- Que han hecho desde la última reunión
- Que tienen planificado hacer antes de la siguiente reunión
- Cualquier obstáculo o impedimento que tengan.

La reunión diaria no es una reunión de estado para informar a un jefe sino más bien sirve para que el equipo auto-organizado pueda compartir entre sus integrantes lo que sucede. Alguien anota los problemas u obstáculos existentes, y el facilitador se encarga de ayudar a los miembros del equipo a resolverlos. Si fuera necesario entrar más en detalle sobre algún problema se hace después de la reunión de scrum, solamente con las personas interesadas.

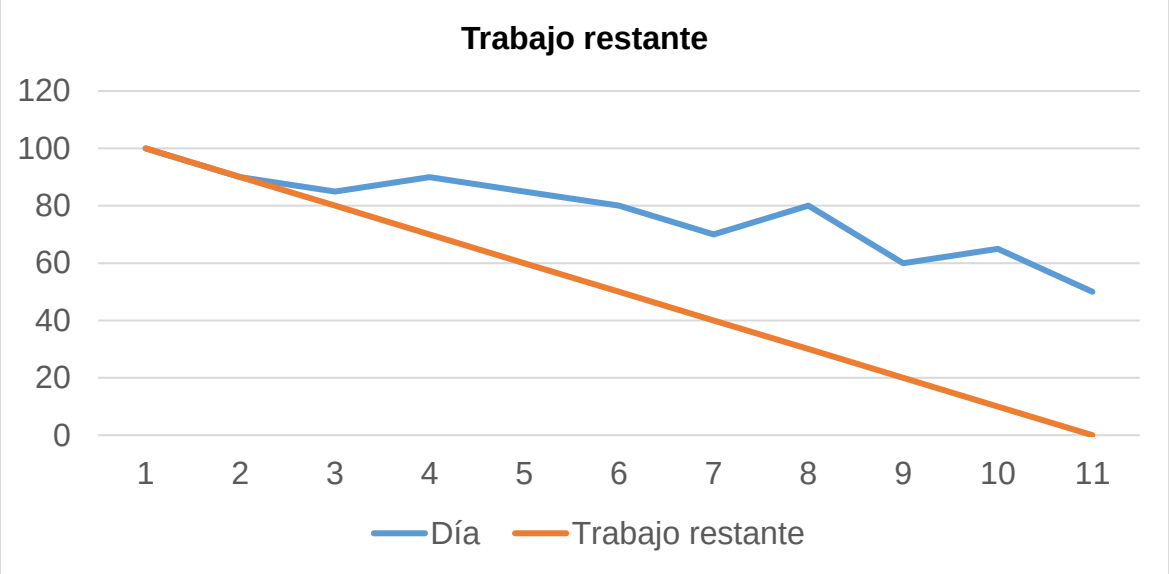
Es aconsejable que los jefes y gerentes no asistan a la reunión de scrum diaria ya que el equipo podría sentirse “observado”. Esto probablemente haría que los integrantes informaran de un gran progreso todos los días o algún resultado no realista, y también podría limitarse la información sobre algún obstáculo del proyecto. (Martel, 2015).

3.5. Actualización de requerimientos de cada fase

Todos los días el equipo actualiza sus estimaciones del trabajo restante y se suman las horas restantes del equipo como un todo, es aconsejable crear una gráfica del trabajo restante del ciclo de trabajo para que el equipo pueda identificar visualmente cuanto trabajo han realizado y el restante, tal como el indicado en la figura 3.

Si la línea de trabajo restante no disminuye conforme avanza el proyecto se deben hacer ajustes, como reducir el alcance del trabajo o encontrar una forma de trabajar más eficientemente. La gráfica debe ser visible por todos los integrantes del equipo, ya sea digital o en un rotulo en la pared. (Martel, 2015)

Figura 3. Trabajo restante de la fase



Fuente: Elaboración propia

3.6. Refinar la lista de requerimientos

Esta actividad de refinamiento se realiza para los elementos de la próxima fase. El equipo debe dedicar del cinco al diez por ciento de cada fase a refinar la lista de requerimientos. Esto incluye realizar el análisis detallado de requerimientos, subdividir los elementos grandes, estimar nuevos elementos, o reestimar los elementos existentes.

Se hace cerca del final de la fase actual, por el equipo y el dueño de producto. Esto hace que la planificación de la fase sea entendida de mejor forma para una estimación más asertiva. El refinamiento es algo breve, no debe incluir muchas preguntas o confusión, ya que esto sería un mal indicador.

3.7. Finalización de la fase

Es importante no prolongar la duración de la fase. El trabajo restante se traslada para la siguiente fase. Alrededor de la tercera o cuarta fase, los equipos son ya

capaces de saber estimar con más precisión y de esta forma se podrá cumplir con los objetivos de la fase con mayor certeza. Una duración estricta de la fase ayuda al equipo a saber cuánto puede hacer, y ayudará en la estimación y la planificación de las próximas entregas.

4. Revisar el ciclo de trabajo

Esta revisión se realiza al finalizar la fase en donde el equipo revisa en conjunto con el dueño de producto, quien puede inspeccionar y adaptar el software; también aprende la forma de uso y da sus comentarios y observaciones. Lo importante de la revisión es una conversación en profundidad entre el equipo y el dueño de producto para conocer la situación y lo que piensa del software realizado en la fase.
El facilitador se encarga de validar la definición de hecho realizada durante la planificación de la fase, y debe decir al dueño de producto si alguno de los elementos implementados por el equipo no cumple esta definición. De esta forma, hay mejor calidad del software y los equipos no pueden simular la calidad al presentar software que parece que funciona bien, pero que está implementado con código desordenado y sin realizar las pruebas correspondientes.

En esta reunión están presentes el dueño de producto, los integrantes del equipo, encargado del equipo y directivos, cualquiera puede realizar preguntas y dar su opinión. (Dimes, 2014).

5. Verificar el ciclo de trabajo

Se inspecciona y adapta el proceso, se realiza después de la revisión, ayuda a obtener una mejora del software en los siguientes ciclos de trabajo, es una oportunidad para que el equipo hable sobre lo que funciona y lo que no, y acuerde los cambios a realizar. Se realiza entre el equipo y el facilitador, opcionalmente también el dueño del producto.

Una forma de estructurar la verificación es, dibujar dos columnas en una pizarra con los textos:

- Qué cosas han funcionado bien.
- Qué se podría mejorar.
- Qué cosas se quiere hacer en la siguiente iteración.
- Cuáles son los problemas que pueden impedir progresar adecuadamente.

Cuando algún elemento se repite, se añade una marca para anotar los elementos comunes y en base a ello se definen los cambios para la siguiente fase, y se compromete a revisar los resultados en la verificada de la próxima fase. En la verificación el equipo analiza cómo ha sido su desempeño durante la fase, si se lograron alcanzar o no los objetivos definidos al inicio de la iteración y si el cliente obtuvo el software esperado. (Dimes, 2014).

6. Fase de entrega

El producto se debe entregar al final de cada fase e incluye un software probado y documentado. El software se puede instalar inmediatamente después de la revisión de la fase.

En caso de haber algún retraso en el desarrollo, se crea una fase especifica solo para la entrega en donde se incluyen las pruebas de integración del entorno de producción; lo ideal, es que ello no sea necesario. El trabajo de la fase de entrega debería ser mínimo si el equipo ha seguido buenas prácticas de desarrollo, con refactorización e integración continua y pruebas efectivas durante cada fase.

7. Planificación de la entrega

En el caso de un producto nuevo o un producto existente que acaba de adoptar Scrum, hay que refinar la lista de requerimientos inicial antes de la primera fase; el

dueño de producto y el equipo dan forma a los requerimientos. Se analizan los requerimientos y realiza la estimación de los elementos identificados para la primera entrega.

En Scrum un producto establecido con una lista de requerimientos no debería necesitar ninguna planificación detallada para la siguiente entrega porque el dueño de producto y el equipo deben refinar la lista de requerimientos en cada fase y preparándose continuamente para el futuro.

Este modo de desarrollo de producto continuo evita la necesidad de las etapas de preparar, ejecutar, concluir, que se ven en desarrollos tradicionales del ciclo de vida secuencial.
En los refinamientos continuos de los requerimientos de cada fase, el equipo y el dueño de producto planifican la entrega, perfeccionan las estimaciones y dan las prioridades y contenido. La atención se centra en crear y perfeccionar un plan para darle una dirección amplia a la entrega y clarificar como se tomarán las decisiones y el alcance que se cubrirá en cada entrega. El dueño de producto acuerda con el equipo estimar los elementos de los requerimientos de entrega que se podrán tener en una fase.

8. Enfocarse en la aplicación

Scrum tiene un modelo de desarrollo continuo del software y no se enfoca en el proyecto en donde hay un inicio y fin; tampoco hay un jefe de proyecto tradicional y en vez de eso hay un dueño de producto estable y un equipo auto-gestionado que trabaja en fases, con una duración de 1 a 4 semanas, hasta que se finaliza la aplicación. El equipo y el dueño del negocio puede ser un cliente interno, hacen toda la gestión del proyecto. No lo gestiona ningún gestor de tecnología o alguien de gestión de proyectos. El equipo se debe centrar en una aplicación durante la fase para obtener un software de mejor calidad. (Rubin, 2013).

RESULTADO 2

Scrum es un conjunto de buenas prácticas para trabajar colaborativamente en equipo y obtener un software de calidad. Estas prácticas tienen su origen en un estudio sobre la manera de trabajar de equipos altamente productivos. El nombre proviene de la comparación entre la forma de trabajo de equipos altamente productivos y multidisciplinarios con la colaboración entre los jugadores de rugby y su formación de Scrum. Es utilizado en proyectos en donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

Esta metodología fomenta el trabajo en equipo, así como la interacción entre las áreas involucradas en los proyectos que se realizan. Los colaboradores adquieren un sentido de responsabilidad más amplio, ya que el equipo de trabajo tiene definido el rol que desempeña y las tareas asignadas.

El trabajo en equipo se divide en los roles siguientes:

- El dueño del producto se encarga de definir los objetivos y trazar la ruta para llegar a ellos; éste, representa al cliente, y define las necesidades del software.
- El facilitador garantiza que los miembros del equipo de trabajo puedan realizar sus tareas adecuadamente; es el guía para que los objetivos se resuelvan a tiempo.
- El equipo Scrum son quienes tienen contacto directo con el software y de quienes depende el progreso de su desarrollo; se puede considerar que son la parte fundamental de toda la estrategia Scrum, ya que de su eficiencia depende la calidad del software para conseguir los resultados esperados.
- Integrantes: forman parte del proyecto, pero desde una perspectiva más alejada y general; usualmente, son directores de diversas áreas, gerentes comerciales, proveedores y supervisores.

La supervisión constante y cíclica del desarrollo del proyecto minimiza riesgos, lo que permite corregir más fácilmente en caso necesario o que se evalúen riesgos futuros que no habían sido contemplados desde el inicio; esto permite tener mayor productividad, calidad y eficiencia. Al tener seguimiento constante, los clientes pueden ver el avance entre cada iteración y asegurarse de que el sistema final cumpla con las características y calidad esperada. Scrum da al proyecto flexibilidad, eficacia y rapidez a los procesos, la reducción de errores y, por tanto, disminuye el gasto en recursos innecesarios.

El equipo se reúne con el cliente para la creación de la lista de requerimientos priorizadas del proyecto, proporciona la estimación de su esfuerzo y aporta sus puntos de vista. En la reunión de planificación de la fase el equipo pregunta al cliente los detalles de los requerimientos para tener una visión del contenido a desarrollar. Al finalizar cada iteración, el equipo realiza una demostración al cliente de los requerimientos completados.

Scrum sistematiza la colaboración dentro del equipo mediante las siguientes actividades:

- Reunión de planificación de la iteración
- Reunión diaria del equipo
- Verificación

Hace que los proyectos sean flexibles al permitir que los requerimientos sean modificables y se utiliza la información conforme el propio avance del proyecto, de forma continua. El software evoluciona en fases con las modificaciones necesarias, hasta que el cliente apruebe la versión final.

RESULTADO 3

De acuerdo al estudio realizado en Scrum se realizan entregas parciales del producto final, priorizadas por el beneficio que aportan al cliente. Se adapta bien a proyectos en entornos complejos, donde se necesita obtener resultados rápidos, en donde los requerimientos son cambiantes o poco definidos. Cada entrega se hace dentro de la fase que es el periodo en donde se realizan todas las acciones pactadas en el documento de requerimientos. Se realizan varias fases hasta que todos los elementos del documento de requerimientos hayan sido entregados. Entre las distintas fases no se deben dejar tiempos improductivos. Para el trabajo pendiente se marcan todas las tareas ya realizadas y también se detalla el estado y la evolución del proyecto para indicar las tareas y requerimientos pendientes.

La integración continua del software realizado, en lapsos cortos de tiempo, ayuda a eliminar rápidamente los obstáculos encontrados, hace que los tiempos de entrega se cumplan y se reducen los riesgos al conocer el tiempo medio que tarda el equipo de desarrollo; también, las entregas funcionales que generan mayor valor para el cliente al inicio del proyecto hacen que el cliente inicie el uso del sistema rápidamente.

Las tareas en Scrum se estiman con el Scrum poker, en donde cada participante tiene una baraja con las siguientes cartas ½, 1, 2, 3, 5, 6, 7 e infinito. En la estimación de una tarea cada participante propone la carta con el número de horas que cree que se necesitan para realizar completamente la tarea. Si la estimación de la tarea resulta ser infinito significa que la tarea ha de ser dividida en varias tareas más pequeñas. Si las estimaciones son muy desiguales, el equipo discute con los participantes que han realizado las estimaciones más alejadas de la media para que justifiquen su elección y aclaren con el dueño de producto el alcance de la tarea con el fin de aproximar las estimaciones con los demás miembros del equipo.
El proceso de estimación implica realizar lo siguiente:

- Cada participante de la reunión tiene un juego de cartas.
- Para cada tarea, historia de usuario o funcionalidad, depende de los requerimientos que se van a estimar, el cliente o dueño del producto expone la descripción y el tiempo máximo.
- Hay establecido otro periodo de tiempo para que el dueño del producto atienda a las posibles preguntas del equipo.
- Cada participarte selecciona la carta, o cartas que representan su estimación, y las separa del resto, boca abajo. Cuando todos han hecho su selección, se muestran boca arriba.
- Si la estimación es infinita, por sobrepasar el límite máximo establecido, la tarea debe dividirse en subtareas de menor tamaño. Si las estimaciones resultan muy desiguales, el equipo debe llegar a un acuerdo con quienes propusieron estás estimaciones. Se solicita al dueño del producto que descomponga la funcionalidad y se valora cada una de las funcionalidades resultantes. Finalmente se llega a un acuerdo para tomar la estimación menor que el equipo considere más apropiada.

Esta técnica hace participar a todos los asistentes, reduce el cuarto de hora o la media hora de tiempo de estimación de una funcionalidad, a escasos minutos, consigue alcanzar consensos sin discusiones y, además hace más dinámica la estimación.

VII. DISCUSIÓN DE RESULTADOS

Se realizó el estudio sobre la administración efectiva de proyectos de software por medio de la metodología Scrum. Para esto se usó las diferentes fuentes bibliográficas referenciadas, que ayudaron a comprender el uso de esta metodología y a tener una visión amplia sobre su utilización.

Se encontró que Scrum es una metodología innovadora en la que se cambia la forma de gestionar tradicionalmente los proyectos de software. Con Scrum se aplica un conjunto de buenas prácticas que fomentan el trabajo en equipo, tiene su origen en el desempeño de grupos altamente productivos y en el juego de rugby, donde todos actúan de común acuerdo para mover la pelota en la cancha.

Conforme al estudio presentado, se considera que Scrum está especialmente indicado, tal como lo menciona el Dr. Sutherland, para proyectos en entornos complejos con requerimientos cambiantes o poco definidos, donde se necesita obtener resultados rápidamente. Para ello, son propuestos tres roles que son dueños de producto quien es el representante del cliente, el equipo quien es el encargado técnicamente de llevar a cabo el proyecto y el facilitador en la gestión del proyecto, ayuda a resolver efectivamente cualquier inconveniente, principalmente administrativo que surja durante su desarrollo.

Se encuentra adecuado definir los requerimientos en historias de usuario, ya que de esta forma los usuarios definen, con un lenguaje entendible, las características que necesitan desarrollar en el sistema; también, es importante hacer notar que pueden sufrir modificaciones para adaptarlas conforme avance el proyecto. La relevancia de la definición de hecho, según (Cohn, 2004), consiste en verificar que se haya dado cumplimiento a los requerimientos aceptados por el cliente. Estos requerimientos se unifican en un listado en los que el cliente define las prioridades y se empiezan a realizar los de mayor valor al inicio del proyecto.

Es conveniente la realización de reuniones diarias ya que esto permite que los integrantes del equipo estén informados del avance del proyecto y de los obstáculos

que se encuentran a cada fase, para buscarles una solución pronta y evitar que el proyecto se atrase. Lo importante de esta reunión es que se informe del avance real para que el equipo pueda reaccionar acorde a la planificación y cumpla con las entregas acordadas.

Se considera de beneficio el hecho de que, previo a la finalización de la entrega de una de las fases del sistema, se empiece a ver lo de la siguiente porque esto permite visualizar más claramente los cambios que deben agregarse en base a lo creado o alguna modificación de lo ya realizado que el cliente considere necesaria y que haya sido observada durante la revisión de la fase.

Se estuvo de acuerdo con que, en las reuniones de revisión, se valide el cumplimiento de los acuerdos suscritos con el cliente y también se apoya que la reunión de verificación de la fase se realice después de la de revisión, ya que el equipo puede su desempeño en el proyecto y ver si alcanzaron los objetivos planteados inicialmente con el cliente. La diferencia, según (Dimes, 2014), es que la revisión se centra principalmente en el cumplimiento con la funcionalidad del sistema.

La relevancia de integrar conforme transcurre cada fase del sistema hasta llegar a la versión final, es que al llegar a la última versión prácticamente ya se tiene todo el sistema requerido por el cliente, se evita tener instalaciones en producción extensiva, lo cual disminuye el tiempo de instalación final, ya que se distribuyó conforme la instalación de cada fase. Se ve factible realizar la gestión de un proyecto de software con la metodología Scrum, dado su modelo de desarrollo continuo en donde no hay jerarquías especificas sino más bien un equipo auto gestionado que trabaja con el fin de obtener un software de calidad.

Es conveniente también para el equipo, el desarrollo en fases incrementales ya que permite al cliente y al equipo técnico colaborar efectivamente en el proyecto simplifica trámites burocráticos lo cual ahorra tiempo de gestión.

De acuerdo al estudio realizado Scrum da flexibilidad, eficacia y rapidez en los procesos y buena comunicación. Esto es posible ya que los requerimientos desde un inicio quedan abiertos a posibles modificaciones que el cliente considera oportunas y también se toma en cuenta las sugerencias del equipo técnico para obtener un sistema óptimo.

Lo propuesto por la metodología Scrum de contar con entregas parciales del sistema final de acuerdo a las prioridades de los clientes ayuda a una buena gestión para acometer proyectos desarrollados en entornos complejos y que necesitan rapidez en los resultados.

El hecho de que el equipo es auto organizado otorga mayor confianza entre los miembros del equipo y hace que la información se comparta fácilmente y genera un ambiente de confianza entre ellos. El equipo, de igual forma, está completamente comprometido con el proyecto y hace todo lo posible para poder entregarlo en la fecha pactada, con los descansos oportunos.

Se estuvo de acuerdo con que el equipo defina con el cliente la creación de la lista de requerimientos de acuerdo a prioridades, ya que esto provee una estimación más efectiva y el equipo puede aportar ideas al cliente que hagan que el sistema sea más funcional y ambas partes están conscientes de lo acordado.

La reunión diaria que el equipo tiene, donde se reúnen aproximadamente por 15 minutos, contribuye a una comunicación más fluida entre el equipo y permite informar del progreso, y actualizar el avance del proyecto.

El hecho de que el equipo revisa y muestra al cliente el software realizado y probado durante cada fase permite al cliente dar sus comentarios y observaciones para incorporarlos en la siguiente fase, una vez tras otra, hasta concluir con el sistema acordado. Esto ayuda a reducir el tiempo de instalación en producción, ya que se logra integrar en cada fase. (Sutherland & K., 2012)

Scrum propone realizar entregas parciales del sistema de acuerdo a las prioridades del cliente con entregas iniciales las que generen más valor para el cliente; conforme el avance del proyecto estas prioridades pueden cambiar.

Prácticamente, la entrega en fases del proyecto final es una de las características que hacen de Scrum una metodología flexible, permite gestionar de mejor forma el proyecto ya que al dividirse en tareas más pequeñas se facilita su estimación. Cada fase entregable del proyecto es conocida en Scrum como fase.

Con este estudio se hizo notar que con Scrum se entrega, de manera regular, avances del sistema al cliente; de esta forma, él puede validar oportunamente cualquier modificación que considere prudente realizar.

El hecho de tener entregas completamente funcionales cada 4 semanas, puede variar de acuerdo a las características del proyecto y lo que haya acordado el equipo. Se miden resultados tempranamente para poder introducir cambios que mejoren el proyecto, lo cual hace más efectivo el uso de los recursos. Es oportuno que, periódicamente, también se evalúe el proyecto conforme su avance, para que cumpla con los estándares aceptables.

La importancia de evitar las interrupciones externas mientras el equipo realize el desarrollo del sistema ayuda a que se mantenga la concentración y la productividad y, de esta forma, se contribuye a la entrega a tiempo de lo acordado.

Hay que resaltar que todo el equipo debe tener conocimiento técnico y trabajar con la misma visión del software que se va a crear; por ello, la importancia de las reuniones de seguimiento y el tamaño del equipo contribuye a que el software se realice en el tiempo estipulado. De acuerdo a la estimación del proyecto, se debe proponer la cantidad de personas que estará involucrada y de esta forma, se pueda cumplir a cabalidad con el tiempo acordado.

VIII. CONCLUSIONES

- Se elaboró un estudio sobre la administración efectiva de proyectos de software mediante la aplicación de la metodología Scrum; se encontró que es posible optimizar el tiempo de creación de un sistema para cumplir con la entrega planificada.

- Se planteó el uso de la metodología Scrum para gestionar efectivamente un proyecto de software, lo cual es factible a través del trabajo en equipo, buena comunicación con los clientes y flexibilidad en la administración.

- Se propuso la realización del sistema final por fases, de acuerdo a las prioridades y especificaciones del cliente, lo que ayuda a estimar de mejor forma el proyecto para cumplir con su entrega a tiempo.

IX. RECOMENDACIONES

- Buscar organizaciones que estén dispuestas a utilizar la metodología Scrum. Es conveniente aplicar esta metodología en empresas en donde exista desorden en el desarrollo de sistemas; o bien, en empresas que inician funciones, es apropiado utilizar esta metodología.

- Iniciar el uso de esta metodología con un proyecto piloto en donde todo el equipo este comprometido y el cliente pueda formar una parte activa. De obtener los resultados esperados, se puede aplicar su uso a los demás proyectos de la empresa.

- Analizar previa y detenidamente los proyectos con que cuenta la empresa y el personal asignado en cada uno, para ver si es factible aplicar la metodología Scrum; antes de iniciar su implementación, explicar al cliente y al equipo los cambios que conlleva su utilización.

X. BIBLIOGRAFÍA

Agile Alliance. (2015). Obtenido del sitio web de Agile Alliance: recuperado el 7 de julio de 2016 de https://www.agilealliance.org/

Alaimo, M., & Salías, M. (2015). *Proyectos ágiles con #Scrum. Flexibilidad, aprendizaje, innovación y colaboración en contextos complejos.* Buenos Aires, Argentina: Kleer.

Albaladejo, X. (2013). Obtenido del sitio web de Proyecto Agiles: recuperado el 7 de julio de 2016 de https://proyectosagiles.org

Anderson, A., Jeffries, R., & Hendrickson, C. (2001). *Extreme Programming Installed.* Estados Unidos de América: Addison-Wesley.

Appelo, J. (2011). *Management 3.0. Leading agile developers, developing agile leaders.* Estados Unidos de América: Addison Wesley.

Applying UML and patterns. (2002). *An introduction to object-oriented analysis and design and the unified process*. United States: Prentice Hall.

Barranco, J. (2001). *Metodología del análisis estructurado de sistemas.* (2 ed.). España: Universidad Pontificia Comillas de Madrid.

Beck, K. (2000). *Extreme Programming Explained. Embrace change.* Estados Unidos de América: Addison-Wesley.

Brechner, E. (2015). *Agile project management with Kanban.* Estados Unidos de América: Microsoft Press.

Camposantos, A. (2014). *Las metodologías ágiles en los proyectos de software.* Estados Unidos: Amazon Media EU.

Canós, J., Letelier, P., & Penadés, M. (2009). *Métodologías Ágiles en el Desarrollo de Software.* Valencia, España: DSIC -Universidad Politécnica de Valencia. Recuperado el 15 de agosto de 2016 de http://www.yises.com/pdfs/agil/TodoAgil.pdf

Cockburn, A. (2004). *Crystal clear. A human powered methodology for small teams.* Estados Unidos de América: Addison Wesley.

Cockburn, A., & Highsmith, J. (2007). *Agile software development. The cooperative game.* (2 ed.). Estados Unidos de América: Addison Wesley.

Cohn, M. (2004). *User stories applied. For agile software development.* Estados Unidos de América: Addison Wesley.

Cohn, M. (2006). *Agile estimating and planning.* Estados Unidos de América: Prentice Hall.

Cortés, R. (2006). *Introducción al análisis de sistemas y la ingeniería de software.* Costa Rica: EUNED.

Cunningham, W. (2001). Obtenido del sitio web de Manifesto: recuperado el 18 de agosto de 2016 de http://agilemanifesto.org/iso/es/manifesto.html

Deemer, P., Benefield, G., Larman, C., & Vodde, B. (2009). *The Scrum Primer.* Estados Unidos de América: Scrum Training Institute.

DeMarco, T., & T., L. (2013). *Peopleware. Productive projects and teams.* (3 ed.). Estados Unidos de América: Addison Wesley.

Derby, E., & Larsen, D. (2006). *Agile retrospectives. Making good teams great.* Estados Unidos de América: Pragmatic Bookshelf.

Dimes, T. (2014). *Scrum Essencial.* Estados Unidos de América: BabelCube, Inc.

Dimes, T. (2015). *Conceptos básicos de Scrum.* España: Babelcube Inc.

Fox, A., & Patterson, D. (2015). *Desarrollando software como servicio. Un enfoque ágil utilizando computación en la nube.* Estados Unidos de América: Strawberry Canyon LLC.

Greene, J., & Stellman, A. (2015). *Learning agile. Understanding Scrum, Xp, Lean, and Kanban.* Estados Unidos de América: O'Reilly Media, Inc.

Gutiérrez, J., & Borillo, R. (2012). *2a Conferencia Agile Spain CAS2011.* Castellón, España: Universitat Jaume I.

Herranz, R. (2016). *Despegar con Scrum.* España: Utópica Informática.

Highsmith, J. (2000). *Adaptive software development. A collaborative approach to managing complex systems.* Nueva York, Estados Unidos de América.: Dorset House, Inco.

Highsmith, J. (2010). Obtenido del sitio web de Jim Highsmith: recuperado el 9 de septiembre de 2016 de http://jimhighsmith.com/

Highsmith, J. (2010). *Agile project managment. Creating innovative products* (2 ed.). Estados Unidos de América: Pearson Education, Inc.

Jacobson, I. (2000). *The road to the unified software development process.* New York, United States: Cambridge University Press.

Keyes, J. (2015). *Extreme Programming Concepts.* Estados Unidos de América: Amazon Digital Services LLC.

Kniberg. (2007). *Scrum and Xp from the trenches. How we do Scrum.* Estados Unidos de América: C4media.

Lacey, M. (2012). *The Scrum field guide. Practical advice for your first year.* Estados Unidos de América: Pearson Education, Inc.

Laínez, J. (2015). *Desarrollo de software ágil. Extremme Programming y Scrum* (2 ed.). Carolina del Sur, Estados Unidos de América: CreateSpace.

Lledó, P. (2012). *Gestión ágil de proyectos.* Estados Unidos: Pablolledó.

Martel, A. (2015). *Gestión práctica de proyectos con Scrum: Desarrollo de software ágil para el Scrum Master* (3 ed.). Carolina del Sur, Estados Unidos de América: CreateSpace.

Maximini, D. (2015). *The Scrum Culture. Introducing agile methods in organizations.* Wendlingen, Alemania: Springer.

Miranda, F., Chamorro, A., & Rubio, S. (2014). *Dirección de operaciones. Casos prácticos y recursos didácticos.* Madrid, España: Paraninfo.

Noriega, R. (2015). *El proceso de desarrollo de software.* Estados Unidos de América: Createspace.

Olivo, A. (2014). *Apuntes sobre Scrum.* Estados Unidos de América: Amazon Media.

Palmer, S., & Felsing, J. (2002). *A practical guide to Feature Driven Development.* Estados Unidos de América: Prentice Hall.

Poppendieck, M., & Poppendieck, T. (2003). *Lean software development. An agile toolkit.* Indiana, Estados Unidos de América: Addison-Wesley.

Pressman, R. (2005). *Software engineering. A practitioner's approach* (6 ed.). Nueva York, Estados Unidos de América: Mc Graw Hill.

Pressman, R. (2010). *Ingeniería del software. Un enfoque práctico.* (7 ed.). México, D.F.: McGraw-Hill.

Proyectos agiles. (2016). Obtenido de Proyectos agiles: https://proyectosagiles.org

Ramos Cardozzo, D. (2016). *Desarrollo de software. Requisitos, estimaciones y análisis* (2 ed.). México. IT Campus Academy.

Ramos, D., Noriega, R., Laínez, J., & Durango, A. (2015). *Curso de ingeniería de software.* Carolina del Sur, Estados Unidos de América: CreateSpace.

Rodríguez, J., García, J., & Lamarca, I. (2007). *Gestión de proyectos informáticos: métodos, herramientas y casos.* Barcelona, España: UOC.

Rubin, K. (2013). *Essential Scrum. A practical guide to the most popular agile process.* Estados Unidos de América: Pearson Education, Inc.

Schenone, M. (2004). *Diseño de una metodología ágil de desarrollo de software.* Buenos Aires, Argentina: Universidad de Buenos Aires.

Schwaber, K. (2004). *Agile project management with Scrum.* Estados Unidos de América: Microsoft Press.

Schwaber, K. (2007). *The enterprise and Scrum.* Redmon, Washington: Microsoft Press.

Schwaber, K., & Beedle, M. (2001). *Agile Software Development with Scrum.* Estados Unidos de América: Prentice Hall.

ScrumOrganization. (2016). Obtenido del sitio web Scrum: recuperado el 9 de septiembre de 2016 de https://www.scrum.org

Sims, C., & Johnson, H. (2011). *The elements of Scrum.* Estados Unidos de América: Dymaxicon.

Sims, C., & Johnson, H. (2012). *Scrum: A breathtakingly brief and agile introduction.* Estados Unidos de América: Dymaxicon.

Sommerville, I. (2005). *Ingeniería del software* (7 ed.). Madrid, España: Pearson Educación, S.A.

Stapleton, J. (1997). *Dynamic Systems Development Method.* Estados Unidos de América: Addison-Wesley.

Stellman, A., & Greene, J. (2006). *Applied software project management.* California, Estados Unidos de América: O'Reilly Media, Inc.

Sutherland, J. (2016). *Scrum. El arte de hacer el doble de trabajo en la mitad de tiempo.* México: Océano.

Sutherland, J., & K., S. (2012). *Software in 30 days.* Nueva Jersey, Estados Unidos de América: John Wiley & Sons, Inc.

Sutherland, J., Solingen, R., & Rustenburg, E. (2011). *The power of Scrum.* Carolina del Sur, Estados Unidos de América: CreateSpace.

Takeuchi, H., & Nonaka, I. (1986). The New New Product Development Game. *Harvard Business Review*, 285-305.

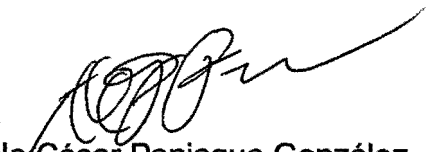
Toro López, F. (2013). *Administración de proyectos de informática.* Bogotá, Colombia: Ecoe Ediciones.

Torossi, G. (2005). *El proceso unificado de desarrollo de software.* Argentina: Universidad Tecnológica.

Watts, G. (2013). *Scrum mastery. From good to great servant leadership.* Estados Unidos de América: Inspect & Adapt Ltd.

Wells, D. (2009). Obtenido del sitio web de Extreme Programming: recuperado el 9 de septiembre de 2016 de http://www.extremeprogramming.org/

Winder, R. (1995). *Desarrollo de software con C++.* Madrid, España: Diaz de Santos, S.A.

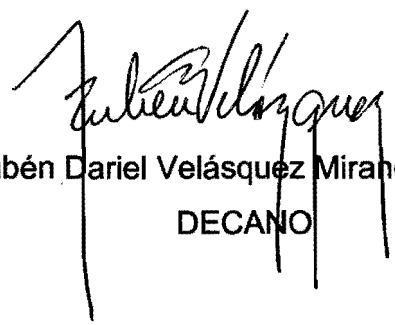


Pablo César Paniagua González

AUTOR

María Ernestina Ardón Quezada, MSc.

DIRECTORA



Rubén Dariel Velásquez Miranda, Ph.D.

DECANO