FORMULACION DE CRITERIOS PARA LA SELECCION DE METODOLOGIAS DE DESARROLLO DE SOFTWARE

LEONARDO FLOREZ MARIN

FELIPE GRISALES TOBON

UNIVERSIDAD TECNOLOGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA EN SISTEMAS Y COMPUTACION

PEREIRA, RISARALDA

2014

CONTENIDO

CAPITULO 1

INTRODUCCIÓN

Con la visión del software como producto nace la necesidad de implementar métodos que garanticen la correcta utilización de los recursos para obtener resultados cada vez más satisfactorios. Dado que el recurso principal para la creación de software es el capital humano[1], se requieren estándares para la unificación de los procesos de ejecución, motivo por el cual nacen las metodologías de desarrollo.

A medida que los avances tecnológicos permiten otras posibilidades en cuanto al software y su desarrollo, se va haciendo evidente la necesidad de la evolución de las metodologías hacia un enfoque que les permita realizar cada vez más trabajo en tiempos más cortos sin ver comprometida la calidad del producto.

Es así como en diversas partes del mundo se comienzan a crear nuevos métodos de desarrollo, con diversas prácticas que agilizan el proceso, cada una con ventajas y desventajas sobre las demás.

Algunas de estas metodologías se mantienen vigentes en la actualidad y son aquellas que reúnen las mejores prácticas dentro de su marco de trabajo. La selección apropiada de una de estas es un proceso complejo, puesto que los proyectos de software son de múltiples naturalezas, tamaños y requerimientos.

El propósito de este documento es el análisis de metodologías más utilizadas a nivel mundial y la búsqueda de parámetros que sirvan como modelo comparativo para la creación de un modelo que facilite la elección de un marco de trabajo para un proyecto determinado.

METODOLOGÍAS DE DESARROLLO TRADICIONALES

Una metodología de desarrollo es un conjunto de prácticas de realización de software que traen como resultado un producto de calidad que satisface las necesidades del cliente. Aquellas que tienen un mayor énfasis en las etapas iniciales del proceso como lo son la formulación de requisitos y el modelado del problema reciben el nombre de metodologías tradicionales[2].

Los modelos de desarrollo más básicos tenían como fundamento el cumplimiento a cabalidad de cada etapa del proceso antes de iniciar la siguiente, lo cual ocasionaba cuellos de botella en las diferentes tapas de la ejecución y prolongaban demasiado los tiempos de entrega.

Con el tiempo estos marcos de trabajo fueron evolucionando en sus prácticas para dar espacio a nuevas formas de desarrollar software, de esa manera se recogen las mejores prácticas de desarrollo dentro de métodos como RUP, el cual recopila todos los procesos requeridos para el correcto funcionamiento de un proyecto de software, mejorando sustancialmente los tiempos de entrega y consiguiendo una reducción en el tiempo transcurrido desde el inicio del proyecto hasta su entrega final.

Para casos prácticos, nos centraremos en especificar las dos metodologías más comunes dentro de los marcos de desarrollo tradicionales. De esta manera, tendremos mayor claridad acerca de sus fortalezas y debilidades.

METODOLOGÍA RUP

Conceptos principales

El proceso unificado es uno de los marcos de trabajo más utilizados en la actualidad para el desarrollo de software. Es orientado a objetos y se guía mediante tres conceptos principales[3]:

Es dirigido por los casos de uso

Los casos de uso son una forma de representar funcionalidades para el usuario, de manera que el cumplimiento de uno de estos dentro del software implica proporcionar un valor agregado al cliente.

La generación de casos de uso va de la mano con el ciclo de desarrollo de software, de manera que estos maduran y avanzan conforme avanza el proceso[3].

Se centra en la arquitectura

La arquitectura dentro de un proyecto de software tiene como objetivo dar forma a cada una de las funcionalidades de los casos de uso, de manera que cada necesidad del usuario se vea cubierta dentro del producto.

A medida que los casos de uso maduran dentro del ciclo de vida del software, también lo hace la arquitectura y viceversa Este proceso requerirá continuar hasta llegar a una estabilidad.

Es iterativo e incremental.

Cada iteración del producto tiene como finalidad ampliar cada vez más la utilidad del software y al mismo tiempo, tratar los riesgos más importantes. De esta manera, se enfocan los equipos de trabajo para producir resultados visibles en tiempos cortos, se disminuyen los costos de los riesgos, dado que se modifican pequeñas partes del software y no en su totalidad.

Estos conceptos son la base del marco de trabajo y se ven apoyados cada uno dentro del otro[3].

FASES

Inicio

Se presentan los casos de uso más críticos, se crea una arquitectura provisional y se hacen estimaciones aproximadas sobre el proyecto, además de realizar una identificación de riesgos[3].

Elaboración

Se hace una revisión profunda de los casos de uso existentes y se elabora la arquitectura sobre la cual se desarrollará el sistema. También se realizará el diseño de una solución preliminar[4].

Construcción

Durante esta fase se crean las funcionalidades del sistema, las funcionalidades se añaden al esqueleto del sistema de manera que este comience a satisfacer cada uno de los requerimientos de software. Al finalizar esta fase el producto debería estar acorde con las necesidades de los usuarios casi en su totalidad[3].

Transición

Se realizan los últimos ajustes al software, corregir los defectos si es necesario, capacitar a los usuarios y la verificación completa de todas las especificaciones expresadas en los requerimientos del sistema[4].

FLUJOS DE TRABAJO

Modelado del negocio

Se pretende lograr un entendimiento completo acerca de la organización para la cual se realizará el proyecto y de igual manera, asegurar esta comprensión por parte de cada uno de los involucrados. Se crea un modelo de casos de uso y un modelo de objetos del negocio.

Requerimientos

Considerado uno de los flujos de trabajo más importantes, se planean las funcionalidades que el sistema debe contener, se realiza un estimado del trabajo a realizar en las iteraciones. La definición de requerimientos funcionales y no funcionales se debe realizar detalladamente y se propone la creación de interfaces gráficas de usuario que faciliten la comprensión y ayuden a delimitar el alcance del proyecto.

Análisis y Diseño

El objetivo es transformar los requerimientos del sistema en un diseño práctico para su futuro desarrollo, teniendo en cuenta también las necesidades no funcionales del cliente. Al final de este flujo se espera obtener un modelo de diseño y la documentación acerca de la arquitectura de software

Implementación

Los diseños de arquitectura se transforman en archivos ejecutables que hacen parte de la estructura del producto, durante cada iteración se establecen los pasos a seguir, las funcionalidades a implementar y su orden específico. Además se realizan pruebas unitarias a los subsistemas y se añaden al software.

Pruebas

Se busca identificar los defectos que pueda poseer el software durante cada iteración, también comprobar que el producto cumpla con las especificaciones acordadas. El propósito final es el refinamiento del producto, planeando y ejecutando las pruebas necesarias.

Despliegue

Se hacen pruebas al software dentro de su entorno real de ejecución, para su posterior instalación y ejecución. También se capacita a los usuarios sobre el funcionamiento de éste, se crean manuales de usuario, asegurando aceptación y adaptación, minimizando de igual manera las complicaciones.

Gestión del proyecto

Mediante un plan para las fases y un plan para las iteraciones se busca gestionar todo lo relacionado con la ejecución del proyecto, de manera que se asegure la consecución de los objetivos mediante documentación, gestión de riesgos, contratación de personal, entre otros.

Configuración y control de cambios

Como su nombre lo indica, este flujo se encarga de controlar las modificaciones que se realizan a los diferentes artefactos creados, llevar registro de dichas modificaciones y asegurando la integridad del proyecto.

Entorno

Se busca proveer información relevante a cada momento, además de entregar herramientas que faciliten el proceso, ayudar con el diseño de los casos de uso, definir las formas de aplicación de cada herramienta de manera adecuada y establecer otras guías sobre los pasos inmediatos a seguir en el proceso[5].

Roles

Analistas

* Analista de procesos de negocio.
* Diseñador del negocio.
* Analista de sistema.
* Especificador de requisitos.

Desarrolladores

* Arquitecto de software.
* Diseñador
* Diseñador de interfaz de usuario
* Diseñador de cápsulas.
* Diseñador de base de datos.
* Implementador.
* Integrador.

Gestores

* Jefe de configuración.
* Jefe de pruebas
* Jefe de despliegue
* Ingeniero de procesos
* Revisor de gestión del proyecto
* Gestor de pruebas.

Apoyo

* Documentador técnico
* Administrador de sistema
* Especialista en herramientas
* Desarrollador de cursos
* Artista gráfico

Especialista en pruebas

* Especialista en Pruebas (tester)
* Analista de pruebas
* Diseñador de pruebas

Otros roles

* Stakeholders
* Revisor
* Coordinación de revisiones
* Revisor técnico

Ventajas

-Reducción de riesgos, dado que reiniciar un ciclo de trabajo no implica empezar de nuevo el proceso de desarrollo, por lo cual el consumo de recursos en caso de posibles eventualidades disminuye.

-Se centra en los casos de uso, facilitando un entendimiento del problema para el equipo desarrollador, promoviendo de esta manera la calidad del software y la adición de valor al cliente del proyecto.

-Permite una visión clara del avance del proyecto en todo momento, permitiendo fijar metas y administrar los recursos disponibles de acuerdo a las necesidades de la metodología.

-Abundante creación de documentos de apoyo que dan información acerca del software.

Desventajas

-Para procesos relativamente pequeños resulta en un trabajo complejo que puede provocar sobrecostos para el desarrollo.

-Una aplicación indebida del modelo puede resultar ineficiente y entorpecer el proceso.

-Los tiempos de entrega son mayores en comparación a metodologías ágiles

-Es una metodología sensible a cambios en etapas avanzadas del proyecto.

RAD

RAD es un proceso de desarrollo de software que surge a mediados de la década de 1970 bajo dirección de Dan Gielan. Este marco de trabajo utiliza un tiempo mínimo de planeación en favor del prototipado rápido. Esto hace la metodología más aplicable a proyectos cortos que se puedan descomponer en partes pequeñas[7].

Metodología

Se centra principalmente en ejecutar los procesos de la manera más rápida posible, con el enfoque de reducir costos e incrementar la calidad. Combina el uso de técnicas de prototipado evolutivo y time boxing[8].

La primera de las etapas de este marco de trabajo es la de planificación de requisitos, en la cual se estructuran las funcionalidades a desarrollar. Cabe resaltar que tales requerimientos deben ser especificados por personal con vasto conocimiento acerca de las necesidades del software[9].

Posteriormente se procede con el diseño de casos de uso, mediante los cuales se asegura el correcto funcionamiento de las funcionalidades del sistema de acuerdo a escenarios prácticos, eliminando así las interfaces ineficaces e inútiles de un desarrollo futuro.

La fase de implementación se ve realizada por equipos pequeños de programación acompañados de usuarios del software, de manera que la retroalimentación sea inmediata. Dado que las pruebas se realizan durante la etapa de construcción, se realizan manuales y algunos documentos que sirven como guía para usuarios finales.

La fase final se centra en la implementación del nuevo producto, de ser necesario, el cambio al nuevo sistema y la capacitación a los usuarios[10].

Ventajas

Se eliminan interfaces no necesarias en etapas tempranas del desarrollo

Los tiempos de entrega son cortos en comparación a otras metodologías

El usuario se ve involucrado durante las etapas de creación del software

El costo de implementación del software se ve reducido.

Desventajas

En proyectos de tamaño grande, el personal necesario para la creación de equipos de trabajo puede exceder los límites presupuestados.

Los prototipos realizados pueden no ser escalables.

Metodologías agiles

Como hemos podido apreciar las metodologías tradicionales o pesadas poseen un marcado énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Sin embargo, este enfoque no resulta ser el más adecuado para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Ante las dificultades para utilizar metodologías tradicionales con estas restricciones de tiempo y flexibilidad, muchos equipos de desarrollo se resignan a prescindir del "buen hacer" de la ingeniería del software, asumiendo el riesgo que ello conlleva.

Para este escenario, las metodologías ágiles emergen como una posible respuesta para llenar ese vacío metodológico. Por estar especialmente orientadas para proyectos pequeños, las metodologías ágiles constituyen una solución a medida para ese entorno, aportando una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto.

Las metodologías ágiles son sin duda uno de los temas recientes en ingeniería del software que están acaparando gran interés y controversia. A mediados de los años 90 comenzó a forjarse una definición moderna de desarrollo ágil del software como una reacción contra las metodologías utilizadas hasta el momento, consideradas excesivamente pesadas y rígidas por su carácter normativo y fuerte dependencia de planificaciones detalladas previas al desarrollo. En el año 2001 diecisiete miembros destacados de la comunidad software (Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland y Dave Thomas), incluyendo algunos de los creadores o impulsores de las metodologías en software, se reunieron en Utah (Estados Unidos) y adoptaron el nombre de "Metodologías ágiles" para denominar a esta nueva corriente de desarrollo. Poco después, algunos de estos miembros formaron la conocida como "Alianza ágil" [11], una organización sin ánimo de lucro que promueve el desarrollo ágil de aplicaciones. Desde ese momento hasta la actualidad las metodologías ágiles han ido adquiriendo gran auge dentro de la industria software y las organizaciones más punteras comienzan a apostar por este nuevo enfoque para desarrollar sus productos.

El desarrollo ágil no especifica unos procesos o métodos que seguir, aunque bien es cierto que han aparecido algunas prácticas asociadas a este movimiento. El desarrollo ágil es más bien una filosofía de desarrollo software. El punto de partida se establece en las ideas emanadas del Manifiesto Ágil tras la reunión de Utah, un

documento que resume la filosofía “agile” estableciendo cuatro valores y doce principios.

Manifiesto ágil

El Manifiesto Ágil comienza enumerando los principales valores del desarrollo ágil. Según el Manifiesto se valora:

• **Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas**. La gente es el principal factor de éxito de un proyecto software. Es más importante construir un buen equipo que construir el entorno. Muchas veces se comete el error de construir primero el entorno y esperar que el equipo se adapte automáticamente. Es mejor crear el equipo y que éste configure su propio entorno de desarrollo en base a sus necesidades.

• **Desarrollar software que funcione por encima una completa documentación.** La regla a seguir es “no producir documentos a menos que sean necesarios de forma inmediata para tomar una decisión importante”. Estos documentos deben ser cortos y centrarse en lo fundamental.

• **La colaboración con el cliente por encima de la negociación contractual.** Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito.

• **Responder a los cambios más que seguir estrictamente un plan.** La habilidad de responder a los cambios que puedan surgir a lo largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc.) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta sino flexible y abierta.

Los valores anteriores inspiran los doce principios del manifiesto. Son características que diferencian un proceso ágil de uno tradicional. Los dos primeros principios son generales y resumen gran parte del espíritu ágil. El resto tienen que ver con el proceso a seguir y con el equipo de desarrollo, en cuanto metas a seguir y organización del mismo.

Estos principios son los siguientes:

I. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.

II. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.

III. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.

IV. La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.

V. Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.

VI. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.

VII. El software que funciona es la medida principal de progreso.

VIII. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.

IX. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.

X. La simplicidad es esencial.

XI. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.

XII. En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.

Estos principios marcan el ciclo de vida de un desarrollo ágil, así como las prácticas y procesos a utilizar [11].

Para el proyecto en curso se consideró utilizar las metodologías más conocidas y mejor documentadas a nivel mundial, estas fueron XP (eXtreme Programming) y SCRUM. A continuación describiremos estas metodologías a fondo para el desarrollo del proyecto.

XP (eXtreme Programming)

Se basa en una serie de reglas y principios que se han ido gestando a lo largo de toda la historia de la ingeniería del software. Usadas conjuntamente proporcionan una nueva metodología de desarrollo software que se puede englobar dentro de las metodologías ligeras, XP está basada en los siguientes valores [12]:

- Simplicidad
- Comunicación
- Retroalimentación
- Coraje o valor
- Respeto.

La idea de XP consiste en traer todo el equipo junto en la presencia de prácticas simples, con suficiente retroalimentación para permitir que el equipo pueda ver dónde está y para acomodar las prácticas a su situación particular [12].

XP cuenta con una serie de actividades para el desarrollo de los proyectos, estas son:

- **Codificar**
- **Hacer pruebas**
- **Escuchar**
- **Diseñar**

Las Historias de Usuario

Las historias de usuario son la técnica utilizada en XP para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento historias de usuario pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas [14].

Roles XP

Aunque en otras fuentes de información aparecen algunas variaciones y extensiones de roles XP, en este apartado enumeraremos los roles de acuerdo con la propuesta original de Beck.

- **Programador**
- **Cliente**
- **Encargado de pruebas (*Tester*)**
- **Encargado de seguimiento (*Tracker*)**
- **Entrenador (*Coach*)**
- **Consultor**
- **Gestor (*Big boss*)**

Proceso XP

Un proyecto XP tiene éxito cuando el cliente selecciona el valor de negocio a implementar basado en la habilidad del equipo para medir la funcionalidad que puede entregar a través del tiempo. El ciclo de desarrollo consiste (a grandes rasgos) en los siguientes pasos [14]:

1. El cliente define el valor de negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
4. El programador construye ese valor de negocio.
5. Vuelve al paso 1.

Prácticas básicas de XP

En la búsqueda de la posibilidad de disminuir la mítica curva exponencial del costo del cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione, XP apuesta por un crecimiento lento del costo del cambio y con un comportamiento asintótico. Esto es posible gracias a las tecnologías disponibles para ayudar en el desarrollo de software y a la aplicación disciplinada de las doce "prácticas básicas" que deben seguirse al pie de la letra [12][13]:

El juego de la planificación

Se hacen las historias de usuario y se planifica en qué orden se van a hacer y las mini-versiones. La planificación se revisa continuamente. El equipo técnico realiza una estimación del esfuerzo requerido para implementar las historias de usuario y los clientes deciden sobre el ámbito y tiempo de los *release* y de cada iteración. El cliente establece la prioridad de cada historia de usuario, de acuerdo con el valor que aporta para el negocio. Los programadores estiman el esfuerzo asociado a cada historia de usuario. Se ordenan las historias de usuario según prioridad y esfuerzo, y se define el contenido del *release* y/o iteración, apostando por enfrentar lo de más valor y riesgo cuanto antes.

Entregas pequeñas

La idea es producir rápidamente mini-versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad pretendida para el sistema, pero que ofrezcan algo útil y de valor para el negocio al usuario final y no trozos de código que no pueda ver funcionando. Deben ser lo suficientemente pequeñas como para poder hacer una cada pocas semanas, una entrega no debería tardar más 3 meses.

Metáfora

El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema. Martin Fowler en [15] explica que la práctica de la metáfora consiste en formar un conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema. Hay que buscar unas frases o nombres que definan cómo funcionan las distintas partes del programa, de forma que sólo con los nombres se pueda uno hacer una idea de qué es lo que hace cada parte del programa.

Diseño simple

Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto. Hacer siempre lo mínimo imprescindible de la forma más sencilla posible. Mantener siempre sencillo el código.

Pruebas

El cliente, con la ayuda de los desarrolladores, propone sus propias pruebas para validar las mini-versiones. La producción de código está dirigida por las pruebas unitarias. Las pruebas unitarias son establecidas antes de escribir el código y son ejecutadas constantemente ante cada modificación del sistema.

Refactorización (*Refactoring*)

El “refactoring” consiste en escribir nuevamente parte del código de un programa, sin cambiar su funcionalidad, a los efectos de hacerlo más simple, conciso y/o entendible.

Programación en parejas

Los programadores trabajan por parejas (dos delante del mismo ordenador) y se intercambian las parejas con frecuencia (un cambio diario). Según Cockburn y Williams en un estudio realizado para identificar los costos y beneficios de la programación en parejas [16], las principales ventajas de introducir este estilo de programación son: muchos errores son detectados conforme son introducidos en el código (inspecciones de código continuas), por consiguiente la tasa de errores del producto final es más baja, los diseños son mejores y el tamaño del código menor (continua discusión de ideas de los programadores), los problemas de programación se resuelven más rápido, se posibilita la transferencia de conocimientos de programación entre los miembros del equipo, varias personas entienden las diferentes partes sistema, los programadores conversan mejorando así el flujo de información y la dinámica del equipo, y por último, los programadores disfrutan más su trabajo.

Propiedad colectiva del código

Cualquier programador puede cambiar cualquier parte del código en cualquier momento. En un proyecto XP, todo el equipo puede contribuir con nuevas ideas que apliquen a cualquier parte del proyecto.

Integración continua

Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día. Es por eso que XP promueve publicar lo antes posible las nuevas versiones, aunque no sean las últimas, siempre que estén libres de errores. Idealmente, todos los días deben existir nuevas versiones publicadas.

40 horas por semana

La metodología XP indica que debe llevarse un ritmo sostenido de trabajo. Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse.

Cliente in-situ

Uno de los requerimientos de XP es tener al cliente disponible durante todo el proyecto. No solamente como apoyo a los desarrolladores, sino formando parte del grupo. Al comienzo del proyecto, el cliente debe proporcionar las historias de usuarios. Pero, dado que estas historias son expresamente cortas y de “alto nivel”, no contienen los detalles necesarios para realizar el desarrollo del código. Estos detalles deben ser proporcionados por el cliente, y discutidos con los desarrolladores, durante la etapa de desarrollo.

Estándares de programación

XP promueve la programación basada en estándares, de manera que sea fácilmente entendible por todo el equipo, y que facilite la recodificación. Se enfatiza la comunicación de los programadores a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación (del equipo, de la organización u otros estándares reconocidos para los lenguajes de programación utilizados).

Ventajas

- Da lugar a una programación sumamente organizada.
- Ocasiona eficiencias en el proceso de planificación y pruebas.
- Cuenta con una tasa de errores muy pequeña.
- Propicia la satisfacción del programador.
- Fomenta la comunicación entre los clientes y los desarrolladores.
- Facilita los cambios.
- Permite ahorrar mucho tiempo y dinero.
- Puede ser aplicada a cualquier lenguaje de programación.
- El cliente tiene el control sobre las prioridades.
- Se hacen pruebas continuas durante el proyecto.
- La XP es mejor utilizada en la implementación de nuevas tecnologías.

Desventajas

- Es recomendable emplearla solo en proyectos a corto plazo.
- En caso de fallar, las comisiones son muy altas.
- Requiere de un rígido ajuste a los principios de XP.
- Puede no siempre ser más fácil que el desarrollo tradicional.

SCRUM

Jeff Sutherland y Ken Schwaber conciben el proceso de Scrum en el principios de los 90. Ellos codificaron Scrum en 1995 con el fin de presentarlo en la conferencia OOPSLA en Austin, Texas (EE.UU.) y publicaron el documento "Proceso de desarrollo de software SCRUM".

Ken y Jeff heredaron el nombre de SCRUM el cual es el término que describe una forma para desarrollar productos iniciada en Japón donde en 1987 Ikujiro Nonaka y Hirotaka Takeuchi usaron este término, una estrategia utilizada en rugby en la que todos los integrantes del equipo actúan juntos para avanzar la pelota y ganar el partido, para denominar un nuevo tipo de proceso de desarrollo de productos. Escogieron este nombre por las similitudes que consideraban que existían entre el juego del rugby y el tipo de proceso que proponían: adaptable, rápido, auto-organizable y con pocos descansos [17].

Scrum es un marco de trabajo por el cual las personas pueden acometer problemas complejos adaptativos, a la vez que entregar productos del máximo valor posible productiva y creativamente. Scrum se caracteriza por ser:

- Ligero
- Fácil de entender
- Extremadamente difícil de llegar a dominar

El marco de trabajo Scrum consiste en los Equipos Scrum, roles, eventos, artefactos y reglas asociadas. Cada componente dentro del marco de trabajo sirve a un propósito específico y es esencial para el éxito de Scrum y para su uso. Las reglas de Scrum relacionan los eventos, roles y artefactos, gobernando las relaciones e interacciones entre ellos.

A continuación analizaremos como funciona SCRUM y que componentes lo conforman.

Equipo SCRUM

El Equipo Scrum consiste en un Dueño de Producto (Product Owner), el Equipo de Desarrollo (Development Team) y un Scrum Master. Los Equipos Scrum son autoorganizados y multifuncionales.

Los Equipos Scrum entregan productos de forma iterativa e incremental, maximizando las oportunidades de obtener retroalimentación. Las entregas

incrementales de producto “Terminado” aseguran que siempre estará disponible una versión potencialmente útil y funcional del producto [18].

Dueño del producto (Product Owner)

El Dueño de Producto es el responsable de maximizar el valor del producto y del trabajo del Equipo de Desarrollo. El Dueño de Producto es la única persona responsable de gestionar la Lista del Producto (Product Backlog). La gestión de la Lista del Producto incluye:

- Expresar claramente los elementos de la Lista del Producto;
- Ordenar los elementos en la Lista del Producto para alcanzar los objetivos y misiones de la mejor manera posible;
- Optimizar el valor del trabajo desempeñado por el Equipo de Desarrollo;
- Asegurar que la Lista del Producto es visible, transparente y clara para todos, y que muestra aquello en lo que el equipo trabajará a continuación; y,
- Asegurar que el Equipo de Desarrollo entiende los elementos de la Lista del Producto al nivel necesario.

Equipo de desarrollo (Development Team)

Construye el producto que va a usar el cliente, por ejemplo una aplicación o un sitio web. El equipo en Scrum es “multi-funcional”, tiene todas las competencias y habilidades necesarias para entregar un producto potencialmente distribuible en cada Sprint, y es “autoorganizado” (auto-gestionado), con un alto grado de autonomía y responsabilidad. En Scrum, los equipos se auto-organizan en vez de ser dirigidos por un jefe de equipo o jefe de proyecto.

Los Equipos de Desarrollo tienen las siguientes características [18]:

- Son autoorganizados. Nadie (ni siquiera el Scrum Master) indica al Equipo de Desarrollo cómo convertir elementos de la Lista del Producto en Incrementos de funcionalidad potencialmente desplegables;
- Los Equipos de Desarrollo son multifuncionales, contando como equipo con todas las habilidades necesarias para crear un Incremento de producto;
- Scrum no reconoce títulos para los miembros de un Equipo de Desarrollo, todos son Desarrolladores, independientemente del trabajo que realice cada persona; no hay excepciones a esta regla;
- Scrum no reconoce sub-equipos en los equipos de desarrollo, no importan los dominios particulares que requieran ser tenidos en cuenta, como pruebas o análisis de negocio; no hay excepciones a esta regla; y,

- Los Miembros individuales del Equipo de Desarrollo pueden tener habilidades especializadas y áreas en las que estén más enfocados, pero la responsabilidad recae en el Equipo de Desarrollo como un todo.

SCRUM Master

El Scrum Master es la persona responsable del éxito al aplicar la metodología SCRUM en el desarrollo del proyecto o producto, asegurando que los valores, prácticas y reglas son seguidos por el resto del equipo. En concreto, es la persona que asegura el seguimiento de la metodología guiando las reuniones, ayudando al equipo ante cualquier problema que pueda aparecer y controlando que el trabajo sigua el ritmo adecuado. Por tanto debe tomar decisiones inmediatas y eliminar los impedimentos que vayan surgiendo en el momento, aunque en ocasiones no cuente con toda la información necesaria. Su responsabilidad es entre otras, la de hacer de paraguas ante las presiones externas y motivar al resto del equipo. Por tanto, la labor de Scrum Master requiere una fuerte personalidad ya que debe facilitar el trabajo del equipo sin imponer autoridad.

Actividades y eventos SCRUM

En Scrum existen eventos predefinidos con el fin de crear regularidad y minimizar la necesidad de reuniones no definidas en Scrum. Todos los eventos son bloques de tiempo (time-boxes), de tal modo que todos tienen una duración máxima. Una vez que comienza un Sprint, su duración es fija y no puede acortarse o alargarse. Los demás eventos pueden terminar siempre que se alcance el objetivo del evento, asegurando que se emplee una cantidad apropiada de tiempo sin permitir desperdicio en el proceso.

Planificación de la iteración (Sprint Planning Meeting)

La planificación de las tareas a realizar en la iteración se divide en dos partes:

Primera parte de la reunión. Se realiza en un timebox de cómo máximo 4 horas:

- El cliente presenta al equipo la lista de requisitos priorizada del producto o proyecto, pone nombre a la meta de la iteración (de manera que ayude a tomar decisiones durante su ejecución) y propone los requisitos más prioritarios a desarrollar en ella.
- El equipo examina la lista, pregunta al cliente las dudas que le surgen, añade más condiciones de satisfacción y selecciona los objetivos/requisitos más

prioritarios que se compromete a completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita.

Segunda parte de la reunión. Se realiza en un timebox de cómo máximo 4 horas. El equipo planifica la iteración, elabora la táctica que le permitirá conseguir el mejor resultado posible con el mínimo esfuerzo. Esta actividad la realiza el equipo dado que ha adquirido un compromiso, es el responsable de organizar su trabajo y es quien mejor conoce cómo realizarlo.

- Define las tareas necesarias para poder completar cada objetivo/requisito, creando la lista de tareas de la iteración (Sprint Backlog) basándose en la definición de completado.
- Realiza una estimación conjunta del esfuerzo necesario para realizar cada tarea.
- Cada miembro del equipo se autoasigna a las tareas que puede realizar.

Sprint

El corazón de Scrum es el Sprint, es un bloque de tiempo (time-box) de un mes o menos durante el cual se crea un incremento de producto “Terminado”, utilizable y potencialmente desplegable. Es más conveniente si la duración de los Sprints es consistente a lo largo del esfuerzo de desarrollo. Cada nuevo Sprint comienza inmediatamente después de la finalización del Sprint previo.

Los Sprints contienen y consisten de la Reunión de Planificación del Sprint (Sprint Planning Meeting), los Scrums Diarios (Daily Scrums), el trabajo de desarrollo, la Revisión del Sprint (Sprint Review), y la Retrospectiva del Sprint (Sprint Retrospective).

Durante el Sprint:

- No se realizan cambios que puedan afectar al Objetivo del Sprint;
- Los objetivos de calidad no disminuyen; y,
- El alcance puede ser clarificado y renegociado entre el Dueño de Producto y el Equipo de Desarrollo a medida que se va aprendiendo más.

Cada Sprint puede considerarse un proyecto con un horizonte no mayor de un mes. Al igual que los proyectos, los Sprints se usan para lograr algo. Cada Sprint tiene una definición de qué se va a construir, un diseño y un plan flexible que guiará la construcción y el trabajo y el producto resultante.

Scrum Diario (Daily Scrum)

El objetivo de esta reunión es facilitar la transferencia de información y la colaboración entre los miembros del equipo para aumentar su productividad, al poner de manifiesto puntos en que se pueden ayudar unos a otros.

Cada miembro del equipo inspecciona el trabajo que el resto está realizando (dependencias entre tareas, progreso hacia el objetivo de la iteración, obstáculos que pueden impedir este objetivo) para al finalizar la reunión poder hacer las adaptaciones necesarias que permitan cumplir con el compromiso conjunto que el equipo adquirió para la iteración. Esto se lleva a cabo inspeccionando el trabajo avanzado desde el último Scrum Diario y haciendo una proyección acerca del trabajo que podría completarse antes del siguiente.

Cada miembro del equipo debe responder las siguientes preguntas en un timebox de cómo máximo 15 minutos:

- ¿Qué he hecho desde la última reunión de sincronización? ¿Pude hacer todo lo que tenía planeado? ¿Cuál fue el problema?
- ¿Qué voy a hacer a partir de este momento?
- ¿Qué impedimentos tengo o voy a tener para cumplir mis compromisos en esta iteración y en el proyecto?

Los Scrum Diarios mejoran la comunicación, eliminan la necesidad de mantener otras reuniones, identifican y eliminan impedimentos relativos al desarrollo, resaltan y promueven la toma de decisiones rápida, y mejoran el nivel de conocimiento del Equipo de Desarrollo. El Scrum Diario constituye una reunión clave de inspección y adaptación.

Revisión de Sprint (Sprint Review)

Reunión informal donde el equipo presenta al cliente los requisitos completados en la iteración, en forma de incremento de producto preparado para ser entregado con el mínimo esfuerzo, haciendo un recorrido por ellos lo más real y cercano posible al objetivo que se pretende cubrir.

En función de los resultados mostrados y de los cambios que haya habido en el contexto del proyecto, el cliente realiza las adaptaciones necesarias de manera objetiva, ya desde la primera iteración, replanificando el proyecto.

El Scrum Master se asegura de que el evento se lleve a cabo y que los asistentes entiendan su propósito. El Scrum Master enseña a todos a mantener el evento dentro del bloque de tiempo fijado.

La Revisión de Sprint incluye los siguientes elementos [18]:

- Los asistentes son el Equipo Scrum y los interesados clave invitados por el Dueño de Producto;
- El Dueño de Producto explica qué elementos de la Lista de Producto se han “Terminado” y cuales no se han “Terminado”;
- El Equipo de Desarrollo habla acerca de qué fue bien durante el Sprint, qué problemas aparecieron y cómo fueron resueltos esos problemas;
- El Equipo de Desarrollo demuestra el trabajo que ha “Terminado” y responde preguntas acerca del Incremento;
- El Dueño de Producto habla acerca de la Lista de Producto en el estado actual. Proyecta fechas de finalización probables en el tiempo basándose en el progreso obtenido hasta la fecha (si es necesario);
- El grupo completo colabora acerca de qué hacer a continuación, de modo que la Revisión del Sprint proporcione información de entrada valiosa para Reuniones de Planificación de Sprints subsiguientes.
- Revisión de cómo el mercado o el uso potencial del producto podría haber cambiado lo que es de más valor para hacer a continuación; y,
- Revisión de la línea de tiempo, presupuesto, capacidades potenciales y mercado para la próxima entrega prevista del producto.

Retrospectiva de Sprint (Sprint Retrospective)

La Retrospectiva de Sprint tiene lugar después de la Revisión de Sprint y antes de la siguiente Reunión de Planificación de Sprint. Se trata de una reunión restringida a un bloque de tiempo de tres horas para Sprints de un mes. Para Sprints más cortos se reserva un tiempo proporcionalmente menor. Con el objetivo de mejorar de manera continua su productividad y la calidad del producto que está desarrollando, el equipo analiza cómo ha sido su manera de trabajar durante la iteración, por qué está consiguiendo o no los objetivos a que se comprometió al inicio de la iteración y por qué el incremento de producto que acaba de demostrar al cliente era lo que él esperaba o no.

El propósito de la Retrospectiva de Sprint es [18]:

- Inspeccionar cómo fue el último Sprint en cuanto a personas, relaciones, procesos y herramientas;
- Identificar y ordenar los elementos más importantes que salieron bien y las posibles mejoras; y,
- Crear un plan para implementar las mejoras a la forma en la que el Equipo Scrum desempeña su trabajo.

Herramientas y artefactos SCRUM

Lista de Producto (Product Backlog)

Es una lista priorizada que define el trabajo que se va a realizar en el proyecto. Cuando un proyecto comienza es muy difícil tener claro todos los requerimientos sobre el producto. Sin embargo, suelen surgir los más importantes que casi siempre son más que suficientes para un Sprint.

Existe un rol asociado con esta lista y es el del cliente o Product Owner. Si alguien quiere realizar cualquier modificación sobre la lista por ejemplo: agregar o incrementar la prioridad de sus elementos tiene que convencer al Product Owner.

Antes de iniciar la primera iteración, el cliente debe tener definida la meta del producto o proyecto y la lista de requisitos creada. No es necesario que la lista sea completa ni que todos los requisitos estén detallados al mismo nivel. En el caso del desarrollo de un producto, la lista va evolucionando durante toda la vida del producto (los requisitos "emergen"). En el caso de un proyecto, conforme éste avance irán apareciendo los requisitos menos prioritarios que falten [19].

Lista de Pendientes del Sprint (Sprint Backlog)

Lista de tareas que el equipo elabora en la reunión de planificación de la iteración (Sprint planning) como plan para completar los objetivos/requisitos seleccionados para la iteración y que se compromete a demostrar al cliente al finalizar la iteración, en forma de incremento de producto preparado para ser entregado.

Esta lista permite ver las tareas donde el equipo está teniendo problemas y no avanza, con lo que le permite tomar decisiones al respecto.

Para cada uno de los objetivos/requisitos se muestran sus tareas, el esfuerzo pendiente para finalizarlas y la autoasignación que han hecho los miembros del equipo.

A partir de los objetivos a cumplir durante el Sprint el Scrum Team determina que tareas debe desempeñar para cumplir el objetivo. De esto surge el Sprint Backlog. Es importante destacar que es el equipo quien se organiza para alcanzar el objetivo. El Manager no asigna tareas a los individuos y tampoco toma decisiones por el equipo. El equipo puede agregar nuevas tareas o remover tareas innecesarias en cualquier momento si lo considera necesario para cumplir el objetivo. Pero el Sprint Backlog solo puede ser modificado por el equipo. Las estimaciones se actualizan cada vez que aparece nueva información [19].

Gráficos de trabajo pendiente (Burndown charts)

En Scrum se planifica y mide el esfuerzo restante necesario para desarrollar el producto. Esta gráfica suele utilizarse en lugar de un diagrama de PERT debido a que el camino crítico en un desarrollo ágil cambia diariamente. Esto haría obsoleto el diagrama de PERT cada día. Es por esto que no es útil una herramienta que modele el camino crítico a partir de actividades. La solución es utilizar una técnica que permita medir la velocidad de desarrollo. Para esto se utiliza el criterio del equipo a partir del cual se calcula diariamente el camino crítico. Esto permite recalcular el plan y la velocidad en que se realiza el trabajo. En función de esto el equipo puede trabajar para acelerar o desacelerar el trabajo para cumplir con la fecha de entrega.

Se pueden utilizan los siguientes gráficos de esfuerzo pendiente:

- Días pendientes para completar los requisitos del producto o proyecto (product burndown chart), realizado a partir de la lista de requisitos priorizada (Product Backlog).
- Horas pendientes para completar las tareas de la iteración (sprint burndown chart), realizado a partir de la lista de tareas de la iteración (Iteration Backlog).

Este tipo de gráfico permite realizar diversas simulaciones: ver cómo se aplazan las fechas de entrega si se le añaden requisitos, ver cómo se avanzan si se le quitan requisitos o se añade otro equipo, etc. [19]

Ventajas:

1. El cliente está satisfecho ya que recibe lo que necesita y esperaba
2. El coste en términos de proceso y Management es mínimo, llevando a un resultado más rápido y barato.
3. Ayuda a la empresa a ahorrar tiempo y dinero
4. Permite realizar proyectos en los que la documentación de los requerimientos de negocios no están muy claros como para ser desarrolladas
5. Se desarrolla rápidamente y testea. Cualquier error puede ser fácilmente rectificado.
6. Los problemas se identifican por adelantado en las reuniones diarias y por lo tanto se pueden resolver rápidamente
7. Hay visibilidad clara del desarrollo del proceso
8. Iterativo en su naturaleza, requiere continuo feedback del usuario
9. Fácil de manejar los cambios debido a los sprints tan cortos y el feedback constante

10. Poco control que insiste en la información frecuente del proceso en el trabajo mediante reuniones regulares.
11. Las reuniones diarias hacen posible medir la productividad individual. Esto lleva a la mejor en la productividad de cada uno de los miembros del equipo.
12. Para los desarrolladores las ventajas pueden ser motivación y satisfacción de realizar el trabajo de una manera eficiente.
13. Es fácil entregar un producto de calidad en el tiempo estipulado
14. Puede trabajar con cualquier tecnología o lenguaje de programación
15. Hace el proceso del desarrollo de software más centrado y manejable

Desventajas:

1. Si no existe una fecha definitiva de finalización del proyecto es posible que se siga solicitando, y añadiendo, nueva funcionalidad.
2. Si una tarea no está bien definida, los costes de tiempo y dinero estimados del proyecto no serán demasiado exactos. En ese caso, la tarea se puede extender sobre varios sprints.
3. Si los miembros del equipo no están centrados y convencidos, el proyecto nunca se completara o incluso fallará.
4. Está bien para proyectos pequeños, de rápido movimiento ya que trabaja bien solo con equipos pequeños.
5. Esta metodología necesita solo miembros de equipo experimentados. Si el equipo consiste en gente que son junior, el proyecto no puede ser completado a tiempo.
6. Además de los recursos sin suficiente experiencia , la falta de dirección firme pueden llevar a los proyectos a no completarse o incluso fallar.
7. La metodología Scrum funciona bien cuando el scrum master confía en el equipo que lleva. Si se practican controles muy estrictos sobre los miembros del equipo, puede ser extremadamente frustrante para ellos, llevando a la desmoralización y el fallo del proyecto.
8. Si algunos de los miembros del equipo se marcha durante el desarrollo puede tener un efecto negativo enorme en el desarrollo del proyecto.
9. El control de la calidad del proyecto es difícil de implementar y cuantificar a menos que el equipo de test puedan llevar a cabo testeo de regresión después de cada sprint.

CAPITULO 2

FORMULACION Y DEFINICIÓN DE CRITERIOS

Dado el análisis realizado en este documento nos encontramos con factores clave que se pueden considerar decisivos a la hora de utilizar una metodología determinada, puesto que las ventajas que ofrece cada una son dependientes de los recursos disponibles para la realización del proyecto y las necesidades futuras de la organización.

Durante el desarrollo de este capítulo analizaremos cada uno de los factores que se pueden considerar críticos dentro de un marco metodológico, tales como presupuesto, personal, entre otros. Sin embargo, se entrarán a considerar otros factores de importancia dentro de los cuales se pueda realizar la elección de un marco de trabajo determinado.

El enfoque de este documento es el análisis de las principales características de los marcos de trabajo tradicionales en comparación con los ágiles, permitiendo de esta manera vislumbrar las diferencias que resaltan en cada uno de ellos y que les otorgan ciertas ventajas bajo condiciones específicas. Dado que las diversas metodologías poseen prácticas y comportamientos diferentes, se utilizan criterios de medición basados en resultados, de esta manera podemos evaluar efectividad, obviando el proceso que cada una realice.

La selección de los marcos de trabajo a evaluar se realizó bajo la premisa de frecuencia de uso, documentación existente y efectividad de la metodología. RUP es el marco de trabajo tradicional más utilizado actualmente y sobre el cual existe la mayor cantidad de documentación. En cuanto a metodologías ágiles, XP y Scrum resaltan por su gran adaptabilidad frente a cambios y sus drásticas reducciones de los tiempos de desarrollo. Si bien el uso de RAD no es común en la actualidad, se seleccionó este marco de trabajo debido a su gran similitud tanto con metodologías tradicionales, como ágiles, permitiendo evaluar la posibilidad de un híbrido.

A continuación se realizará la discriminación de criterios que servirán como base para una futura calificación cuantitativa, de manera que sea claramente visible la

diferencia en el uso de una u otra metodología de desarrollo. Se pretende resaltar las diversas ventajas y falencias de los marcos de trabajo analizados en cuanto al criterio seleccionado.

2.1 Presupuesto disponible

A la hora de llevar a cabo un proyecto es de vital importancia realizar un estimado del presupuesto que se va a destinar a este. Los costos de implementación de cada metodología varían, dados los requerimientos específicos que cada una de ellas posee.

Teniendo en cuenta las investigaciones hechas en los anteriores capítulos del proyecto podemos observar los diferentes presupuestos que cada metodología puede tener por los diferentes recursos, artefactos y personal que éstas requieren para su desarrollo.

RUP puede llegar a ser la metodología más costosa, dependiendo del tamaño del proyecto. RAD puede implicar costos muy bajos para un producto de buena calidad. Por otro lado tanto XP como Scrum por ser metodologías ágiles no demandan muchos gastos en cuestiones de personal y recursos para el desarrollo de los proyectos.

2.2 Tamaño del proyecto

Las metodologías tradicionales van enfocadas principalmente hacia proyectos grandes que conlleven desarrollos a largo plazo. RUP es una metodología pesada, orientada a los casos de uso y con estándares que facilitan el desarrollo ordenado para proyectos grandes, sin embargo, en proyectos relativamente pequeños puede ocasionar algunos sobrecostos. RAD es una metodología cuya efectividad se ve reducida en proyectos de tamaño demasiado grande. XP y Scrum están orientadas principalmente a proyectos no demasiado extensos

2.3 Tiempos limitados de entrega

Todo proyecto, independiente de su tamaño, se ve sujeto a limitaciones de tiempo, las cuales pueden llegar a marcar la diferencia entre la selección de una metodología ágil o una tradicional. Las metodologías ágiles se caracterizan por tener tiempos cortos de diseño e implementación por sus cortas iteraciones. En contraposición, las metodologías tradicionales poseen una mejor organización a la hora de la división del trabajo, conllevando iteraciones más prolongadas. RAD es

una metodología diseñada para realizar entregas viables en tiempos relativamente cortos. RUP requiere una cantidad mayor de tiempo para sus iteraciones, en comparación con una metodología ágil.

2.4 Necesidad de documentación

Para diversos equipos de trabajo, dependiendo de su tamaño y organización, se hace necesaria la creación de documentos con una mayor o menor profundidad. No todas las empresas requieren documentación exhaustiva sobre su software o los procesos para llevarlo a cabo. La creación de manuales de usuario es opcional dentro de algunas empresas.

Tanto XP como Scrum carecen del manejo de una documentación formal para el desarrollo de los proyectos, la única documentación que estas 2 metodologías ofrecen es el código resultado de las diferentes iteraciones. RUP es una metodología orientada a la creación de múltiples documentos de apoyo para los diversos procesos.

2.5 Personal necesario

Existen diferentes tamaños de proyecto, cada uno con sus requerimientos de personal, dado el software y hardware necesario, la necesidad de un equipo interdisciplinario y la coordinación requerida entre cada área del desarrollo.

XP cuenta con numerosos roles para el control de los procesos en la diferentes iteraciones y el número de personas puede aumentar dependiendo del tamaño del grupo de programadores, aun así el total de miembros no sobrepasa los 15. Scrum es la metodología que menos personal necesita ya que posee muy pocos roles y su tamaño crece dependiendo del grupo de programación que puede ir de 5 a 10 personas. RUP es una metodología con roles definidos para grupos grandes de programadores.

2.6 Adaptabilidad, Respuesta a cambios

No todo proyecto se ve sujeto a cambios repentinos durante su planeación, pero todos deberían poseer un mecanismo de respuesta ante estos. La posibilidad de que ocurra un cambio repentino varía de acuerdo al tipo de proyecto y sus cualidades.

XP y Scrum como metodologías ágiles responden a uno de los valores en los cuales son basados y esto es la flexibilidad que poseen en respuesta a los cambios que

se pueden presentar en el desarrollo del proyecto. Las metodologías tradicionales como RUP son sensibles a cambios, principalmente en etapas avanzadas del proyecto.

2.7 Imposibilidad del cliente

El cliente es la parte más importante en el desarrollo de cualquier proyecto de software, dado que es él quien provee todos los requerimientos, especificaciones y detalles del proyecto que se va a llevar a cabo, por lo tanto la disponibilidad de tiempo del cliente para el proyecto es un tema a poner en consideración.

Scrum como metodología ágil tiene al cliente como parte del equipo llamándolo Product Owner, el cual participa constantemente de las correcciones y observaciones en las diferentes entregas de los Sprints. XP como parte de su estructura de desarrollo posee en sus prácticas como fundamental la participación del cliente no solamente como apoyo a los desarrolladores, sino formando parte del grupo. RUP es un modelo incremental e iterativo, de manera que la interacción con el cliente se realiza principalmente en etapas tempranas de planeación y desarrollo.

CAPITULO 3

CUANTIFICACIÓN DE CRITERIOS

Teniendo en cuenta los criterios analizados en el capítulo anterior, buscamos generar valores que nos permitan cuantificar la eficacia de cada marco de trabajo para situaciones específicas. Procedemos así a asignar valores a cada metodología según su grado de cumplimiento con los criterios existentes.

La evaluación será realizada utilizando números enteros en el rango de (1,5), asignando un valor objetivo, dependiendo directamente de la necesidad específica de cada proyecto.

Los valores asignados reflejarán la aplicabilidad de la metodología en cuestión a dicho caso en particular, siendo (5) el valor que indica el mejor cumplimiento del criterio analizado respecto a las prácticas utilizadas dentro del marco de trabajo. En caso contrario, un valor de (1) implica que tales prácticas resultan contraproducentes para el correcto desarrollo del proyecto dentro de las pautas propuestas.

En la tabla 1 se puede observar la asignación de tales valores, de manera que se puedan referenciar fácilmente las metodologías y sus ventajas en ciertos factores que pueden considerarse decisivos a la hora de tomar decisiones.

Esta tabla es una de las entradas del modelo matricial de selección que se describirá en el siguiente capítulo.

	RUP	RAD	XP	SCRUM
Presupuesto disponible	**1**	**3**	**4**	**5**
Tamaño del proyecto	**5**	**4**	**2**	**1**
Tiempos limitados de entrega	**1**	**2**	**4**	**5**
Necesidad de documentación	**5**	**4**	**3**	**1**
Personal necesario	**5**	**3**	**2**	**1**
Adaptabilidad, respuesta a cambios	**1**	**2**	**4**	**5**
Imposibilidad del cliente	**4**	**3**	**1**	**2**

Tabla 1. Criterios cuantitativos de selección

CAPITULO 4

PROPUESTA DEL MODELO EVALUATIVO

Habiendo identificado los puntos críticos de cada marco de trabajo, enumerado y cuantificado las áreas críticas de los proyectos en general y teniendo un modelo de evaluación para cada una, procedemos entonces a describir un modelo de selección que facilite la toma de decisiones acerca de las metodologías a utilizar.

El modelo que aquí se plantea parte de una selección binaria que pretende resaltar las fortalezas de las diferentes metodologías, de manera que la selección de estas sea la más adecuada posible según la visión general que el dueño del proyecto posea.

La tabla 2, mostrada a continuación, establece una serie de preguntas que sirven como guía para la asignación de valores binarios que posteriormente se utilizarán para realizar la evaluación de la tabla 1.

Condiciones Evaluativas	Si(1) - No(0)
¿Posee limitaciones de presupuesto para el desarrollo del proyecto?	
¿Puede el proyecto considerarse de tamaño grande?	
¿Es necesario que el desarrollo del software se realice en un periodo corto de tiempo en relación con el tamaño de este?	
¿Se requiere un volumen amplio de documentación en las diferentes etapas del proyecto?	
¿El proyecto requiere ser desarrollado por un equipo amplio y multidisciplinario?	
¿Considera que el proyecto a realizar es susceptible a diversos cambios durante su ejecución?	
¿Existe alguna imposibilidad del cliente de estar presente durante todo el proceso de desarrollo del proyecto?	

Tabla 2. Valores evaluativos.

Descripción del modelo

A continuación se realiza la descripción general del proceso planteado:

Evaluación de necesidades básicas del proyecto

El dueño del producto o líder de proyecto, mediante el uso de la tabla de Valores evaluativos define, según su criterio, cuáles son los puntos críticos para su proyecto asignando valores de 0 o 1, dependiendo de los recursos y necesidades que este tenga.

Producto de matrices de las tablas definidas

Después de obtener los datos del líder de proyecto, se realizará un producto de matrices entre los resultados de la tabla de Valores evaluativos y los elementos numéricos de la tabla de los Criterios cuantitativos definida en el capítulo 3.

Sumatoria de valores

Con el resultado del producto de matrices entre las tablas procedemos a sumar los valores obtenidos de este procedimiento, columna a columna, el cual nos entregará los valores obtenidos para cada metodología bajo los criterios establecidos por el dueño del producto.

Obtención de resultados

Con los resultados obtenidos definimos por orden ascendente, cuál ha sido la metodología que más satisfactoriamente cumple con los valores de negocio estipulados por el dueño del proyecto. De esta manera, la metodología con el valor más alto, será aquella más adecuada para llevar a cabo la ejecución del proyecto evaluado

CAPITULO 5

CASO DE APLICACIÓN

Para mejorar la comprensión de lo planteado en el presente proyecto, procederemos a evaluar el proceso dentro de un posible escenario.

Necesidades del dueño del producto

Para el inicio de la evaluación definiremos de una manera muy simple los requerimientos del dueño del producto, donde evaluaremos las necesidades en base a los criterios definidos dentro del proyecto y los plantearemos en la tabla de Valores evaluativos.

Para este ejemplo tenemos un líder de proyecto, el cual desea implementar un software para la gestión de una tienda web, con las siguientes necesidades:

- Se cuenta con el presupuesto justo para la realización del proyecto.
- La base de datos no sobrepasa los 50000 usuarios.
- Posee un tiempo limitado para la realización del proyecto, aproximadamente 3 meses.
- No requiere mucha documentación, puesto que solo está interesado el funcionamiento de la base de datos, sin embargo no requiere manuales explicativos de las interfaces.
- Dispone de un grupo pequeño de programadores y analistas para el desarrollo del proyecto.
- No tiene definidas a totalidad las funcionalidades del sistema, por lo cual el proyecto puede estar sometido a cambios en las etapas siguientes del desarrollo.

- El cliente que solicitó la creación de la base de datos solamente dispuso de unos requerimientos en una reunión inicial, ya que por motivo de viaje no estará acompañando la realización del proyecto.

Una vez obtenidas las definiciones básicas del proyecto procedemos con la interpretación de la información en nuestra tabla:

Condiciones Evaluativas	**Si(1) - No(0)**
¿Posee limitaciones de presupuesto para el desarrollo del proyecto?	1
¿Puede el proyecto considerarse de tamaño grande?	0
¿Es necesario que el desarrollo del software se realice en un periodo corto de tiempo en relación con el tamaño de este?	1
¿Se requiere un volumen amplio de documentación en las diferentes etapas del proyecto?	0
¿El proyecto requiere ser desarrollado por un equipo amplio y multidisciplinario?	0
¿Considera que el proyecto a realizar es susceptible a diversos cambios durante su ejecución?	1
¿Existe alguna imposibilidad del cliente de estar presente durante todo el proceso de desarrollo del proyecto?	1

Producto de matrices

Ahora que tenemos los datos del dueño del producto podemos proceder con el cálculo del producto entre las tablas y obtener las respectivas valoraciones de las metodologías en base a los criterios definidos en la tabla anterior.

	RUP	RAD	XP	SCRUM	Producto	Evaluación
Presupuesto disponible	**1**	**3**	**4**	**5**	**x**	**1**
Tamaño del proyecto	**5**	**4**	**2**	**1**	**x**	**0**
Tiempos limitados de entrega	**1**	**2**	**4**	**5**	**x**	**1**
Necesidad de documentación	**5**	**4**	**3**	**1**	**x**	**0**
Personal necesario	**5**	**3**	**2**	**1**	**x**	**0**
Adaptabilidad, respuesta a cambios	**1**	**2**	**4**	**5**	**x**	**1**
Imposibilidad del cliente	**4**	**3**	**1**	**2**	**x**	**1**

5.3 Sumatoria de valores

Procedemos hacer una sumatoria columna a columna del resultado dado por el producto del punto anterior, lo cual nos deja la tabla de la siguiente manera:

	RUP	RAD	XP	SCRUM
Presupuesto disponible	**1**	**3**	**4**	**5**
Tamaño del proyecto	**0**	**0**	**0**	**0**
Tiempos limitados de entrega	**1**	**2**	**4**	**5**
Necesidad de documentación	**0**	**0**	**0**	**0**
Personal necesario	**0**	**0**	**0**	**0**
Adaptabilidad, respuesta a cambios	**1**	**2**	**4**	**5**
Imposibilidad del cliente	**4**	**3**	**1**	**2**
TOTAL	**7**	**10**	**13**	**17**

5.4 Obtención de resultados

Como podemos observar de la tabla anterior, estos son los resultados de las metodologías para el caso dado:

RUP: 7

RAD: 10

XP: 13

SCRUM: 17

De estos resultados podemos analizar:

- Scrum resalta como la metodología más apropiada para el desarrollo del presente proyecto.
- XP es una posible segunda opción en caso tal que la metodología arrojada en el proceso presente dificultades para el grupo de desarrollo.

CAPITULO 6

CONCLUSIONES

A lo largo de este documento se realizó un análisis cuantitativo acerca de las diferentes metodologías de desarrollo de software, teniendo como resultado un modelo de selección basado en resultados.

De este proceso podemos deducir que el marco de trabajo bajo el que se desarrolla RAD resulta ser cercano a un híbrido entre las metodologías tradicionales y las metodologías ágiles, desenvolviéndose bien ante diferentes situaciones, sin llegar a resaltar en algún sentido.

Si bien el proceso planteado se puede tomar como una referencia para la selección de un marco de trabajo, cabe resaltar que existen otros factores influyentes al momento de tomar dicha decisión. Tales factores pueden ser necesidades específicas no contempladas dentro de los criterios, dado que los proyectos tecnológicos poseen cualidades que son inherentes al entorno sobre el cual se desarrollan.

Es importante tener en cuenta que la selección de una metodología es un proceso que se debe llevar a cabo por personal capacitado, puesto que no todas ofrecen los mismos resultados bajo idénticas condiciones y en cualquier proyecto se cuenta con recursos limitados cuyo aprovechamiento resulta conveniente maximizar.

BIBLIOGRAFIA

[1]LABORATORIO nacional de calidad del software, Ingeniería del software: Metodologías y ciclos de vida [en línea], [Fecha de consulta: 12/10/2014]. Disponible en: https://www.incibe.es/file/N85W1ZWFHifRgUc_oY8_Xg

[2] TICONA, Shirley Fabiola, Metodologías tradicionales, metodologías ágiles, metodologías para juegos, metodologías educativas y metodologías para aplicaciones móviles [en línea], [Fecha de consulta: 11/10/2014]. Disponible en: http://tallerinf281.wikispaces.com/file/view/METODOLOG%C3%8DAS+TRADICIONALES.pdf

[3]JACOBSON, Ivar, GRADY, Booch y RUMBAUGH, James, El proceso unificado de desarrollo de software [en línea], [Fecha de consulta: 12/10/2014]. Disponible en: http://unpprogespcn.com/material/RUP/ElProcesoUnificadodeDesarrollodeSoftware.pdf

[4] ZARAGOZA, María de Lourdes, Desarrollando aplicaciones informáticas con el Proceso de Desarrollo Unificado (RUP) [en línea], [Fecha de consulta: 14/10/2014]. Disponible en: http://www.utvm.edu.mx/OrganoInformativo/orgJul07/RUP.htm

[5]JOYA, Javier Enrique, Flujos de trabajo (metodología RUP) [en línea], [Fecha de consulta: 14/10/2014]. Disponible en: http://javi-adsi.blogspot.com/2009/10/flujos-de-trabajo-metodologia-rup.html

[6] RAMIREZ, Eglis Airana, Metodología RUP [en línea], [Fecha de consulta: 13/10/2014]. Disponible en: https://eglisramirez6.wordpress.com/conociendo-tecnologias/

[7]STICKYMINDS, Function Point Analysis and Agile Methodology [en línea], [Fecha de consulta: 14/10/2014]. Disponible en: http://www.stickyminds.com/article/function-point-analysis-and-agile-methodology

[8]CASEMAKER Inc, What is Rapid Aplication Development? [en línea], [Fecha de consulta: 04/10/2014]. Disponible en: http://www.casemaker.com/download/products/totem/rad_wp.pdf

[9]CASTRO, José Fernando, METODOLOGÍA DE DESARROLLO DE SOFTWARE RAD [en línea], [Fecha de consulta: 15/10/2014]. Disponible en: http://metodologiarad.weebly.com/uploads/1/5/6/7/15678332/exposicionmetodologadedesarrollodesoftwarerad-100928194319-phpapp01.pptx

[10]ARIES, Benjamín, Ciclo de vida del modelo de desarrollo RAD [en línea], [Fecha de consulta: 04/10/2014]. Disponible en: http://www.ehowenespanol.com/ciclo-vida-del-modelo-desarrollo-rad-como_548102/

[11] AGILE Alliance [en línea], [Fecha de consulta: 04/10/2014]. Disponible en: www.agilealliance.org/the-alliance/

[12] WELLS, Don, Extreme Programming: A gentle introduction [en línea], [Fecha de consulta: 04/10/2014]. Disponible en: http://www.extremeprogramming.org

[13] BECK, Kent, Extreme Programming Explained. Embrace Change, Pearson Education [en línea], [Fecha de consulta: 06/10/2014]. Disponible en: http://books.google.es/books?id=G8EL4H4vf7UC&printsec=frontcover&hl=es#v=onepage&q&f=false

[14] JEFFRIES, Ron, ANDERSON, Ann, HENDRICKSON, Chet, Extreme Programming Installed [en línea], [Fecha de consulta: 07/10/2014]. Disponible en: http://homes.di.unimi.it/~belletc/corsi/xpinstall.pdf

[15] FOWLER, Martin, Is Design Dead? [en línea], [Fecha de consulta: 09/10/2014]. Disponible en: www.martinfowler.com/articles/designDead.html

[16] COCKBUN, Alistair, WILLIAMS, Laurie, The Costs and Benefits of Pair Programming [en línea], [Fecha de consulta: 14/10/2014]. Disponible en: http://dsc.ufcg.edu.br/~jacques/cursos/map/recursos/XPSardinia.pdf

[17] SCRUM guides The History of Scrum [en línea], [Fecha de consulta: 15/10/2014]. Disponible en: http://www.scrumguides.org/history.html

[18] SCHWABER, Ken, SUTHERLAND, Jeff, The SCRUM guide [en línea], [Fecha de consulta: 16/10/2014]. Disponible en: http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf

[19] KNIBERG, Henrik, SCRUM and XP from the trenches [en línea], [Fecha de consulta: 18/10/2014]. Disponible en: http://wwwis.win.tue.nl/2R690/doc/ScrumAndXpFromTheTrenchesonline07-31.pdf