risti
Revista Ibérica de Sistemas e Tecnologias de Informação
Iberian Journal of Information Systems and Technologies

Recebido/Submission: 13/05/2021
Aceitação/Acceptance: 29/07/2021

Modelo de Gestión de Desarrollo de Software Ágil mediante Scrum y Kanban sobre la Programación Extrema

Hubner Janampa Patilla[1], Edgar Gómez Enciso[2], José Carlos Juárez Pulache[1], Jorge Luis Lozano Rodríguez[1], Eder Solórzano Huallanca[1], Yudith Meneses Conislla[1]

hubner.janampa@unsch.edu.pe; edgar.gomez3@unmsm.edu.pe; jose.juarez@unsch.edu.pe; jorge.lozano@unsch.edu.pe; eder.solorzano.27@unsch.edu.pe; yudith.meneses@unsch.edu.pe

[1] Unidad de Investigación e Innovación FIMGC, Universidad Nacional de San Cristóbal de Huamanga, Ayacucho, 05002, Huamanga, Perú.

[2] Universidad Nacional Mayor de San Marcos, Lima, 150121, Perú.

Pages: 450-466

Resumen: Para desarrollar software, es menester, la elección de una metodología. En este contexto aparecen las metodologías ágiles. Este enfoque muestra su efectividad en proyectos con requisitos muy cambiantes, asimismo permite reducir los tiempos de desarrollo, pero manteniendo una alta calidad. Entre las metodologías agiles más populares tenemos: Programación Extrema (XP), Scrum y Kanban. El primero está enfocado en la parte de la programación, el segundo está enfocado en las prácticas de organización y gestión, el tercero gestiona un óptimo flujo de trabajo dentro del proceso, en consecuencia, estas metodologías pueden complementarse y obtener un producto de mejor calidad. El objetivo de esta de investigación es implementar el modelo de desarrollo de software de la Programación Extrema sobre Scrum y Kanban para permitir la gestión de software ágil. Se extrae conceptos relevantes de la Programación Extrema, Scrum y Kanban, para presentar modelo hibrido que permitirá desarrollar software ágil.

Palabras-clave: Programación Extrema sobre Scrum y Kanban; gestión de software ágil; metodologías agiles; modelo híbrido.

Agile Software Development Management Model using Scrum and Kanban on Extreme Programming

Abstract: To develop software, it is necessary to choose a methodology. Agile methodologies appear in this context. This approach shows its effectiveness in projects with very changing requirements, it also allows reducing development times, but maintaining high quality. Among the most popular agile methodologies

we have: Extreme Programming (XP), Scrum and Kanban. The first is focused on the programming part, the second is focused on organization and management practices, the third manages an optimal work flow within the process, consequently, these methodologies can be complemented and obtain a better quality product. The objective of this research is to implement the Extreme Programming software development model on Scrum and Kanban to allow agile software management. Relevant concepts are extracted from Extreme Programming, Scrum and Kanban, to present a hybrid model that will allow the development agile software.

Keywords: Extreme Programming on Scrum and Kanban; agile software management; agile methodologies; hybrid model.

1. Introducción

Para desarrollar aplicaciones, es necesario seleccionar una metodología de desarrollo de software. En la actualidad, las metodologías agiles son las más utilizadas, sin embargo, la problemática surge cuando necesitamos una metodología ágil para gestionar el proceso de desarrollo de software y al mismo tiempo que se utilice las prácticas de programación; con la metodología Scrum, una metodología ágil para gestionar el proceso de desarrollo de software, lo integramos con la metodología ágil de la Programación Extrema, y finalmente incorporar a Kanban para la gestión de un flujo óptimo de trabajo dentro del marco del proceso propuesto.

Se logró: a) Proponer un modelo de desarrollo de software iterativo e incremental de la Programación Extrema sobre Scrum y Kanban que permita la gestión de software ágil a través de la planificación del Sprint. b) Se propone un modelo de desarrollo de software a través de las pruebas unitarias continuas de la Programación Extrema sobre Scrum y Kanban para permitir la gestión de software ágil a través de la revisión del Sprint. c) Se propone un modelo de desarrollo de software a través de la refactorización del código de la Programación Extrema sobre Scrum y Kanban que permita la gestión de software ágil a través de la retrospectiva del Sprint. d) Se propone un modelo de desarrollo de software a través de la corrección de errores y código compartido de la Programación Extrema sobre Scrum y Kanban que permita la gestión de software ágil a través del Scrum diario.

2. Metodología

2.1. Estrategia de Integración

Se consolidará las mejores prácticas de XP que está más enfocado a las técnicas de programación y pruebas del software, combinándolo con Scrum que está más enfocado a las prácticas de gestión y organización, Kanban más enfocado en gestionar de manera general como se van completando las tareas, y en la mejora continua, adaptándose como la base del nuevo marco de trabajo. En base a estas premisas se procede a generar un modelo de integración de las metodologías XP, Scrum y Kanban expresadas en las siguientes matrices (tablas del 1 al 8):

	XP	SCRUM	KANBAN	NOTA
ROLES	*Cliente*	*Product Owner*	*Service Request Manager*	*Similares funciones*
	Encargado de seguimiento	*Scrum Master*	*Service Delivery Manager*	*Similares funciones*
	Programador	*Scrum Team*		*Similares funciones*
	Encargado de pruebas			*Adoptado en la propuesta de modelo de gestión ágil*

Tabla 1 – Modelo de integración de las metodologías XP, Scrum y Kanban (Roles)

	XP	SCRUM	KANBAN	NOTA
ACTIVIDADES Y PROCESOS	*Planeación* *Análisis* *Diseño*	*Product Backlog (pila del producto)* *Sprint Planning*	*Concientización de la metodología*	*Similares funciones* *Complementar con el análisis de requerimientos* *Diseño simple fácilmente entendible que ayudará en la comunicación y desarrollo* *Adoptado en la nueva propuesta*
	Codificación	*Sprint*	*Implementar priorizando componentes con más dificultades* *Implementar el resto de componentes*	
	Pruebas unitarias *Pruebas de aceptación* *Pruebas de integración*			
		Scrum diario *Sprint Review Meeting* *Retrospectiva* *Re Release Planning Meeting*	*Revisión del sistema Kanban*	*Adoptado en esta nueva propuesta de modelo de gestión de trabajo ágil*

Tabla 2 – Modelo de integración de las metodologías XP, Scrum y Kanban (Actividades y procesos)

	XP	SCRUM	KANBAN	NOTA
ARTEFACTOS	Historias de usuario	Product Backlog (pila del producto)		Combinadas unas con otras para mejorar las especificaciones del cliente
	Tarjetas CRC	Sprint Backlog		
	Task Cards			
		Burn down chart	Tablero Kanban	
		Burn up chart	Cartas Kanban	

Tabla 3 – Modelo de integración de las metodologías XP, Scrum y Kanban (Artefactos)

	XP	SCRUM	KANBAN	NOTA
VALORES	Comunicación	Comunicación	Transparencia	Similitud conceptual
			Colaboración	
			Entendimiento	
	Retroalimentación	Retroalimentación	Equilibrio	
			Enfoque al cliente	
	Sencillez		Flujo	
	Valentía	Responsabilidad		
	Respeto		Liderazgo	
			Acuerdo	
			Respeto	

Tabla 4 – Modelo de integración de las metodologías XP, Scrum y Kanban (Valores)

	XP	SCRUM	KANBAN	NOTA
PRACTICAS	Cliente in situ	Cliente presente en todo el proceso		Combinar unas con otras para poder mejorar pruebas, y estandarizar código.
	Semana de 40 horas	Iteración de 2 a 4 semanas	Visualizar el flujo de trabajo	
	Metáfora		Gestionar el flujo	
	Diseño simple		Políticas explícitas	
	Re factoring		Retroalimentación	
	Programación en parejas		Eliminar interrupciones	
	Liberaciones cortas	Liberaciones cortas		
	Pruebas			
	Código estándar			
	Integración continua	Integración continua		
	Propiedad colectiva			
	Juego de planificación			

Tabla 5 – Modelo de integración de las metodologías Xp, Scrum y Kanban (Practicas)

De esta manera el nuevo marco de trabajo “XP + Scrum + Kanban”, propuesto se da en las siguientes matrices:

	XP+SCRUM+KANBAN
ROLES	Product Owner
	Scrum Master
	Scrum Team
	Encargado de pruebas

Tabla 6 – Nuevo marco de trabajo "XP + Scrum + Kanban" – Roles

	XP+SCRUM+KANBAN			NOTA
ACTIVIDADES Y PROCESO	Planeación			Concientizar nueva metodología
	Análisis			
	Diseño	Políticas explícitas	Sprint Planning	
		Diseño simple		
	Codificación		Sprint	Implementación de todos los componentes
	Pruebas	Pruebas unitarias		
		Pruebas de aceptación		
		Pruebas de integración		
	Scrum diario			Revisión del sistema
	Sprint Review Meeting			
	Retrospectiva			
	Re Release Planning Meeting			

Tabla 7 – Nuevo marco de trabajo "XP + Scrum + Kanban" – Actividades y Procesos

	XP+SCRUM+KANBAN		
ARTEFACTOS	Historias de Usuario	Product Backlog	Tablero Kanban
	Tarjetas CRC	Sprint Backlog	
	Task Cards		Cartas Kanban
	Burn down chart		
	Burn up chart		

	XP+SCRUM+KANBAN	
VALORES	*Comunicación*	
	Enfoque al cliente	
	Sencillez	
	Responsabilidad	
	Retroalimentación	
PRÁCTICAS TÉCNICAS	*Iteración de 2 a 4 semanas*	*Limitar el WIP*
	Visualizar el flujo de trabajo	
	Liberaciones cortas	
	Programación en parejas	
	Refactoring	
	Código estándar	
	Integración continua	
	Propiedad colectiva	

Tabla 8 – Nuevo marco de trabajo "XP + Scrum + Kanban" – Artefactos, Valores y Prácticas Técnicas

3. Resultados

Producto de la integración de las metodologías ágiles analizadas se obtuvo el siguiente esquema, figura 1:

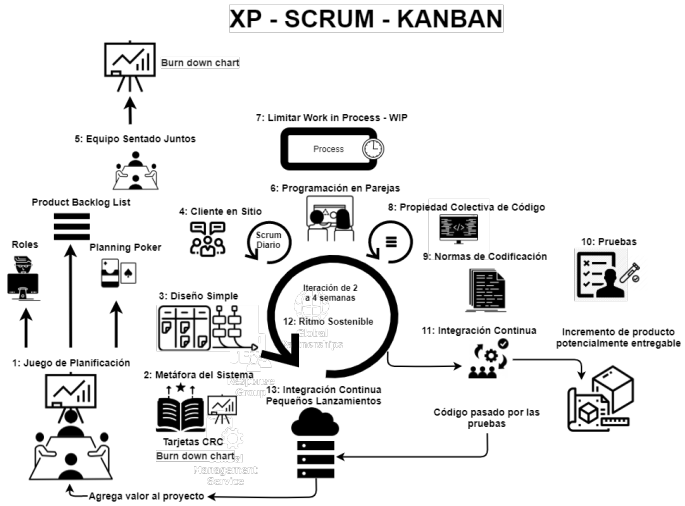

Figura 1 – Marco de trabajo "XP + Scrum + Kanban"

Visto el marco propuesto a través de la figura 1, definiremos cada una de ellas.

3.1. Juego de planificación

Se define el trabajo que se va a realizar en el proyecto, se especifican funciones generales, es decir aquellas que tienen mayor importancia y pueden ser realizadas en un periodo de tiempo fijo. "Al comienzo de cada Sprint se hace la Reunión de Planificación del Sprint. Se divide en dos reuniones" (Deemer, Benefield, Larman, & Bas, 2009).

Los miembros del equipo discuten sobre requisitos con el propietario del producto y revisan los criterios de aceptación para asegurarse del común entendimiento de lo que esperan (Cifuentes Lozano, 2014).

Los proyectos XP generalmente se realizan en iteraciones de 2 semanas, entregando código listo para el cliente al final de cada iteración. El equipo se reúne con el cliente en el primer día de una iteración, identifica las características que se implementarán en la siguiente iteración y presenta los resultados al cliente al final de la iteración (Stober & Hansman, 2010).

3.2. Planning Poker

Técnica ágil de estimación y planificación basada en el consenso. El proceso de planificación del póker se repite hasta que se logre el consenso o hasta que los estimadores decidan una ágil estimación y planificación de un artefacto, en particular debe diferirse hasta que se pueda adquirir información adicional (Cohn, 2006). A través del análisis de 101 estimaciones de historias de usuarios, realizadas por un equipo de XP para la planificación de lanzamientos, investigamos que la introducción del proceso de estimación de la planificación del póker mejora la capacidad de estimación del equipo. Los resultados muestran que la planificación del póker mejoró el rendimiento de la estimación del equipo en la mayoría de los casos, pero que aumentó el error de estimación en los casos extremos (Haugen, 2006).

3.3. Metáfora del Sistema

Usar glosarios de términos y una correcta especificación de nombres de los métodos y clases, ayudará a comprender el diseño, y facilitará sus posteriores ampliaciones, así como la reutilización del código. Debe ser una explicación coherente del sistema que se descompone en trozos más pequeños de historias. Las historias siempre deben expresarse en el vocabulario de la metáfora, y el lenguaje de la metáfora debe ser común tanto para el cliente como para los desarrolladores (Dooley, 2017, p.17).

Los equipos comienzan a descomponer la lista de requerimientos seleccionados en tareas, que para entregarlas deben estar completas. En esta parte se definen las tareas necesarias para complementar cada objetivo o requisito creando la lista de tareas de la iteración (Sprint Backlog) basándose en la definición de hecho (Cifuentes Lozano, 2014). En esta actividad se realizan las siguientes sub actividades: detallar las tareas, determinar responsabilidades, identificar los riesgos, estimación del tiempo de cada miembro del equipo, crear el tablero de tareas – Task Board, crear la figura de producto – Burn Up Chart.

Permite conocer al Product Owner las versiones previstas, las funcionalidades de cada una, velocidad estimada, fechas previstas, margen error y velocidad real. (Palacio, 2007).

3.4. Tarjetas CRC (Clase Responsabilidad Colaboración)

Se presentan sus responsabilidades en la parte izquierda de la tarjeta y en la parte derecha la lista de los colaboradores de la clase. Luego en el estadio de diseño avanzado o en la implementación del sistema, las tarjetas CRC se convierten en clases con métodos, atributos, relaciones de herencia, composición o dependencia (Casas & Reinaga, 2009).

3.5. Diseño Simple

Un diseño simple se implementa más rápidamente que uno complejo. Se sugiere nunca adelantar la implementación de funcionalidades que no correspondan a la iteración en la que se esté trabajando (Joskowicz, 2008).

Se diseña el código de la manera más sencilla posible para cumplir con los requisitos de la historia actual. Usan el diseño más simple que satisface los requisitos inmediatos (Koch, 2005, p.178).

3.6. Kanban

Kanban es una herramienta para aprender y gestionar un óptimo flujo de trabajo dentro del proceso (Klipp, 2014).

3.7. Tablero Kanban

Según (Kanbanize, 2020), es una gran herramienta de ayuda que sirve para mejorar la eficiencia del flujo de trabajo porque visualiza todas las tareas en un proceso de trabajo y proporciona una transparencia general de la organización, permitiendo a los equipos tener una clara visión de todos los elementos de trabajo y controlarlos a través de las diferentes etapas de su flujo de trabajo. Las cartas Kanban ayudan en la identificación de las tareas que estaban por hacer, las que se están haciendo y las que ya se hicieron.

3.8. Cliente en el sitio

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 (Letelier & Penadés, 2006).

Un cliente es parte del equipo, está en el sitio, escribe y ejecuta pruebas funcionales y ayuda a aclarar los requisitos. La capacidad del cliente para dar la retroalimentación inmediata a los cambios en el sistema también aumenta la confianza del equipo en que están construyendo el sistema correcto todos los días (Dooley, 2017, p.18).

3.9. Equipo Sentado Juntos

“El entorno físico debe ser un ambiente que permita la comunicación y colaboración entre todos los miembros del equipo durante todo el tiempo, cualquier resistencia del cliente o del equipo de desarrollo hacia las prácticas y principios puede llevar el

proceso al fracaso (el clima de trabajo, la colaboración y la relación contractual son claves), el uso de tecnologías que no tengan un ciclo rápido de realimentación o que no soporten fácilmente el cambio, etc." (Letelier & Penadés, Repositorio institucional de la Universidad de Las Tunas, 2012).

El motivo de esta práctica es garantizar la facilidad para compartir la información. Esta práctica también mantiene a todos en el equipo comprometidos mientras trabajan para garantizar una carga de trabajo equilibrada (Canty, p.40).

3.10. Marco de trabajo Scrum

Según PMI (2017) es un marco de proceso que se utiliza para gestionar el desarrollo de productos. El marco consta de roles, artefactos y reglas de Scrum, utiliza un enfoque iterativo para entregar un producto funcional. Scrum se ejecuta en timeboxes de un mes o menos con duraciones consistentes llamados sprints donde se produce un incremento de producto potencialmente liberable (p.101).

Según Dimes (2015) es un marco de referencia dentro de la metodología de desarrollo de software ágil, el cual lo habilitará para crear software excelente, mediante la aplicación de un conjunto de directrices a seguir por los equipos de trabajo y el uso de roles concretos.

Scrum se puede describir mejor como una metodología de desarrollo de producto con aspiraciones de gestión de proyectos leves, ya que se centra en la gestión de los requisitos de software y desarrollo (Mora, 2015, p.19).

Los roles del marco de trabajo Scrum, son tres:

El propietario del producto (Product Owner). Schwaber y Sutherland (2017) señalan que el dueño de producto es el responsable de maximizar el valor del producto y del trabajo del equipo de desarrollo. El cómo se lleva a cabo esto podría variar ampliamente entre distintas organizaciones, equipos Scrum e individuos. El dueño de producto es la única persona responsable de gestionar el Product Backlog (p.6).

Scrum Master. Es el responsable del cumplimiento de las reglas del marco de Scrum. Se asegura que éstas son entendidas por la organización y de que se trabaja conforme a ellas. Asesora y da la formación necesaria al propietario del producto y al equipo, y configura, diseña y mejora de forma continua las prácticas ágiles de la organización (Palacio, 2019).

Equipo de desarrollo (Development Team). Según Schwaber y Sutherland (2017), consiste en los profesionales que desempeñan el trabajo de entregar un incremento de producto "terminado", que potencialmente se pueda poner en producción, al final de cada sprint. Solo los miembros del equipo de desarrollo participan en la creación del incremento. Los miembros del equipo de desarrollo son estructurados y empoderados por la organización para organizar y gestionar su propio trabajo. La sinergia resultante optimiza la eficiencia y efectividad del Equipo de Desarrollo (p.7).

Según Rubin (2013), las actividades de Scrum son los siguientes:

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 (Schwaber y Sutherland, 2017, p. 9).

Sprint Planning. Durante la planificación del sprint, el propietario del producto y el equipo de desarrollo acuerdan un objetivo de sprint que define lo que se supone que alcanzará el próximo sprint. Usando este objetivo el equipo de desarrollo revisan el backlog de producto y determina los elementos de alta prioridad que el equipo puede lograr de manera realista en el próximo sprint trabajando a un ritmo sostenible, un ritmo al que el equipo de desarrollo pueda trabajar cómodamente por un periodo de tiempo (Rubín, 2013, p.21).

Scrum diario. Las reuniones diarias para Scrum, son "conversaciones" de no más de 5-15 minutos, que el Scrum master tendrá al comienzo de cada día, con cada miembro del equipo. En esta conversación, el Scrum Master deberá ponerse al día de lo que cada miembro ha desarrollado (en la jornada previa), lo que hará en la fecha actual, pero, sobre todo, conocer cuáles impedimentos estén surgiendo, a fin de resolverlos y que el Scrum team pueda continuar sus labores, sin preocupaciones (Bahit, 2012, p.52).

Ejecución del Sprint. La ejecución del sprint es el trabajo que realiza el equipo Scrum para cumplir con el objetivo del sprint. Durante la ejecución del sprint, los miembros del equipo de desarrollo realizan activamente trabajo creativo de diseño, construcción, integración y prueba de elementos de la lista de productos en incrementos de funcionalidad. Para hacer esto, se auto organizan y deciden como planificar, administrar y comunicar el trabajo. El equipo de desarrollo pasa la mayor parte de su tiempo realizando ejecución de sprint (Rubin, 2013, p.347).

Revisión del sprint. Afirma que el objetivo de esta actividad es inspeccionar y adaptar el producto que se está construyendo. Crítico para esta actividad es la conversación que tiene lugar entres sus participantes, que incluyen el equipo Scrum, stakeholders, patrocinadores, clientes y miembros interesados de otros equipos. La conversación se centra en revisar las características que se acaban de completar en el contexto del esfuerzo de desarrollo general. Todos los asistentes obtienen una visibilidad clara de lo que está ocurriendo y tiene la oportunidad de ayudar a guiar el próximo desarrollo para garantizar la solución más adecuada para el negocio (Rubin, 2013, p.26).

Retrospectiva del sprint. Esta actividad ocurre con frecuencia después de la revisión del sprint y antes de la próxima planificación del sprint. Durante la retrospectiva del Sprint el equipo de desarrollo, Scrum master y el propietario del producto se unen para discutir que funciona y que no funciona con Scrum y las prácticas técnicas asociadas. La retrospectiva es una oportunidad para inspeccionar y adaptar el proceso. Al final de una retrospectiva de sprint, el equipo Scrum debería tener identificado y comprometido un número práctico de acciones de mejora de procesos que será llevado a cabo por el equipo Scrum en el próximo sprint (Rubin, 2013, p.27). En la retrospectiva del sprint a nivel de proceso de gestión y organización, planteamos los siguientes ejercicios.

1. **Cronología de eventos.** Crear una línea de tiempo del evento es una forma simple pero poderosa de generar un artefacto compartido que representa visualmente el flujo de eventos durante un sprint (Rubin, 2013).

2. **Sismógrafo de emociones.** El sismógrafo de emociones es un complemento del ejercicio de cronología del evento. Esta es una representación gráfica de los altibajos emocionales del participante en el transcurso del sprint (Rubin, 2013).

Según Rubin (2013), los artefactos de Scrum son los siguientes:

Product backlog. Es una lista ordenada de todo lo que podría ser necesario en el producto, y es la única fuente de requisitos para cualquier cambio a realizarse en el producto. La lista de producto enumera todas las características, funcionalidades, requisitos, mejoras y correcciones que constituyen cambios a ser hechos sobre el producto para entregas futuras. Los elementos de la lista de producto tienen como atributos la descripción, la ordenación, la estimación y el valor (Schwaber y Sutherland, 2017, p. 15).

Sprint backlog. El Sprint Backlog consiste en los ítems del backlog del producto pronosticados por el equipo de desarrollo en el sprint actual. También contiene el desglose de tareas para mostrar cómo convertir estos ítems en un incremento de producto "terminado". El equipo de desarrollo lo crea durante la reunión de planificación de sprint y es el único responsable de eso. Las personas ajenas al equipo no tienen voz en su creación o uso. El Sprint Backlog es un artefacto vivo, que potencialmente se refina, actualiza y corrige cada día. Siempre muestra el estado actual del sprint. El sprint backlog nunca puede usarse como una herramienta para aplicar presión sobre el equipo de desarrollo (Maximini, 2015, p.299).

Incremento. El incremento es la parte de producto producida en un sprint, y tiene como características que está completamente terminada y operativa en condiciones de ser entregada al cliente final (Palacio, 2020).

3.11. Fase de preparación (Sprint cero).

Es un periodo preparatorio a lo largo del cual, según los casos, se establecen los elementos técnicos necesarios para el proyecto (Subra y Vannieuwenhuyze, 2018, p.124). Asimismo, durante este periodo se debe escribir las historias de usuario que serán estimados y priorizados para su implementación en la reunión de planificación del sprint, estos artefactos son relevantes como expresa Cohn (2009), una historia de usuario describe la funcionalidad que será valiosa para el usuario u comprador de un sistema software (p.4). En consecuencia, debe cumplir ciertas características desde el punto de vista de Wake (2003), los criterios que definen los atributos de una historia de usuario efectiva son identificados con el acrónimo INVEST.

3.12. Programación en pareja

Dos desarrolladores trabajan en equipo en una computadora en la misma tarea, ya que de los dos programadores se quedará uno que tendrá el conocimiento suficiente sobre lo que estaba haciendo la pareja como para poder seguir adelante. Sin embargo, también en este caso, las condiciones que se dan en el desarrollo de software libre hacen que esto no sea tan problemático (Robles & Ferrer, 2002). La programación en parejas generalmente requiere algo de práctica para ser realmente efectiva y puede parecer un poco extraño para los desarrolladores al principio (Stober y Hansman, 2010, p.49).

3.13. Limitar el WIP

La mejor opción es optar por un límite alto e irlo bajando a cada dos o tres semanas hasta encontrar el equilibrio (Bermejo, 2011).

3.14. Propiedad colectiva de código

La propiedad del código compartido promueve que todo el personal pueda involucrarse para corregir y ampliar cualquier parte del código, además, todo lo anterior en conjunto con las pruebas frecuentes garantizan que los errores sean detectados de manera temprana (Balza, Briceño, Lobo, & Nuñez, 2017).

El equipo es dueño de todo, lo que implica que cualquiera puede cambiar cualquier cosa en cualquier momento. En algunos lugares, esto se conoce como "programación sin ego". Los programadores deben aceptar la idea de que cualquiera puede cambiar su código y que la propiedad colectiva se extiende desde el código a todo el proyecto; esto es un proyecto de equipo, no uno individual (Dooley, 2017, p.17). El código compartido establece que todos los miembros del equipo que descubren que algo anda mal en el sistema tienen derecho a corregir cualquier parte del código en cualquier momento, esta práctica requiere que todos actúen con responsabilidad (Janes & Succi, 2014, p.91).

3.15. Estándares de codificación

Entendemos como estándar de código a un conjunto de convenciones establecidas de ante mano (denominaciones, formatos, etc.) para la escritura de código. Estos estándares varían dependiendo del lenguaje de programación elegido y además varían en cobertura, algunos son más extensos que otros (ohmyroot, 2017).

Para practicar la propiedad colectiva del código en un proyecto XP, es necesario que todos los desarrolladores sigan un código estándar para asegurarse de que el código se vea como si una persona hubiera creado cambios. No se requiere un código estándar especifico, sin embargo, todos los desarrolladores deben ser consistentes con el método de escritura del código (Canty, 2015, p.39).

3.16. Pruebas

La prueba de aceptación del usuario también se deriva de criterios de aceptación y se automatizará tanto como sea posible. El desarrollo basado en pruebas requiere que el equipo escriba las pruebas antes de desarrollar el código. La idea detrás del desarrollo basado en pruebas es que el ciclo de pruebas y retroalimentación es lo más corto posible para que la retroalimentación pueda ocurrir temprano (Canty, 2015, p.39).

3.17. Refactoring

Las refactorizaciones son ampliamente reconocidas como formas de mejorar la estructura interna del software orientado a objetos mientras se mantiene su comportamiento externo. (Du Bois, Demeyer, & Verelst, 2004).

Un cambio realizado en la estructura interna del software para que sea más fácil de entender y más barato de modificar sin cambiar su comportamiento observable (Fowler, 2019, p.56).

3.18. Integración continúa

En este marco la integración continua ofrece un esquema que permite realizar integraciones a medida que se lleva a cabo el desarrollo, generando incrementos pequeños y mostrando los resultados obtenidos (Salamon, y otros, 2014).

Esta práctica se refiere a la integración del código y asegura que todo el código funcione bien junto. Esta práctica es importante porque los problemas se pueden detectar rápidamente antes de integrar un código defectuoso o un diseño no coincidente. La integración continua garantiza que las pruebas de integración se realicen en el código cada vez que se agrega una nueva funcionalidad. Esto da como resultado problemas detectados inmediatamente (Canty,2015, p.39).

3.19. Ritmo sostenible

Planificar el trabajo para mantener un ritmo constante y razonable, sin sobrecargar al equipo (Rosas, 2017). XP entiende que el equipo debe mantener un nivel constante de productividad. Esto solo se puede lograr si el equipo mantiene un ritmo sostenible que se traduce en 40 horas semanales de trabajo (Canty, 2015, p.39).

3.20. Re Release Planning Meeting

Reunión con el cliente para revisar y replantear las prioridades del proyecto. El problema de la planificación de lanzamiento (RP) puede investigarse desde dos dimensiones: qué liberar y cuándo liberar. Una desventaja importante de los métodos de RP existentes es que no consideran los sistemas existentes al tomar decisiones de RP (Saliu & Ruhe, 2007).

3.21. Retrospectiva

Última ceremonia de un Sprint. Oportunidad para el equipo de inspeccionarse a sí mismo, y crear un plan de mejora para el siguiente Sprint (Tesei, Cabrera, Vaquero, & Tedini, 2019).

3.22. Pequeños lanzamientos

Como anteriormente se ha comentado, las metodologías ágiles priorizan las entregas y/o animan a realizar pequeños entregables de manera que haya una evaluación continua y constante. En Scrum estas entregas se controlan a través de iteraciones llamadas sprints (Moré Martín, 2011). En XP se obliga a que los incrementos sean lo más corto posible limitando el contenido de cada lanzamiento. Se supone que cada lanzamiento demuestra una pequeña cosa que tiene valor para el cliente. Debido a esta regla, el cliente tiene muchas oportunidades para ver el progreso del proyecto y guiar al equipo en la construcción del producto adecuado (Koch, 2005, p.110).

4. Conclusiones

Los puntos de mejora que se han visto beneficiados con este marco de trabajo ágil pasan a ser descritos a continuación:

Se ha logrado que el equipo de trabajo esté enfocado en un determinado proyecto y no en varios a la vez, esto permite que el equipo reaccione y se ajuste a las necesidades de un solo proyecto y esté totalmente concentrado en este trabajo a tiempo completo. Se crearon equipos multidisciplinarios y auto organizados con una visión clara de roles y responsabilidades, haciendo que el trabajo en equipo se convierta en una herramienta eficiente.

Al trabajar con el esquema de iteraciones cortas y entregas de código funcional, esto permite en gran medida que los proyectos sean entregados a tiempo y con calidad. Las historias de usuario usadas permiten tener una forma documentada de lo que se va a ir creando además de darle al usuario la posibilidad de priorizar la importancia de las mismas, así como también ayudaron a que los miembros del equipo se comprometan en la entrega de resultados finales. De igual manera con la entrega de software funcional en cada iteración y con la revisión y pruebas realizadas se puede medir la satisfacción del cliente. Al tener reuniones periódicas con el equipo de trabajo y los usuarios se logró que los miembros del equipo de desarrollo tengan una visión mucho más amplia del proyecto. Otro de los puntos que se ha visto mejorado es la elección del trabajo por parte de los miembros del equipo de acuerdo a sus aptitudes. El hecho de llevar una plantilla de lecciones aprendidas es de gran ayuda puesto que permite registrar errores y soluciones que bien documentadas puede ayudar a otros equipos a no caer en las mismas problemáticas.

La reunión de retrospectiva permitió ayudar al equipo a ver las deficiencias que se dieron en el trabajo de las iteraciones y ver cómo solucionarlos para los futuros sprint. Las expectativas del cliente en las iteraciones han sido resueltas en la mayoría de los casos, se han mantenido en constante comunicación con el mismo y se lo ha involucrado en el proyecto de una forma muy activa tanto en el levantamiento de los requerimientos como en las pruebas del mismo. La entrega de resultados anticipados, puesto que de esta manera el cliente está en posición de comenzar a usar partes de su producto antes de que el proyecto se encuentre finalizado, de esta forma se puede ir midiendo el progreso que va teniendo el proyecto.

Al usar las mejores prácticas de XP, Scrum y Kanban se ha logrado:

La creación de software con calidad mediante la aplicación de las correspondientes pruebas unitarias, de concepto y de aceptación, se ha logrado que en la primera iteración el producto entregable final sea de un alto grado de calidad, adaptado totalmente a las necesidades del cliente, y probado por el mismo cliente. El uso de tarjetas CRC han dado una gran ventaja y ayuda al momento de levantar las historias de usuario de cada uno de los requerimientos. Definitivamente un gran aporte ha sido el tener una codificación mucho más simple y entendible con lo que se reduce el número de errores gracias al uso del refactoring.

El uso de la práctica de programación en pares, si bien es cierto que para muchas personas puede resultar tediosa, ha sido de gran aporte, ya que ha permitido a los programadores aplicar buenas prácticas al momento de codificar, probar y controlar el código. Los usos por separado de estas metodologías hubieran dado buenos resultados, pero, por ejemplo, si el proyecto se hubiese realizado estrictamente con XP, la calidad de las aplicaciones tendría un mayor grado, la propiedad colectiva del código se manejaría

de mejor manera, la programación en pares hubiera permitido trabajar de mejor forma a los programadores, pero la gestión de desarrollo y con el cliente pudiera ser menos eficaz, lo que ocasionaría que existan muchos reprocesos. Si se hubiese usado únicamente Scrum en el proyecto, se hubiera podido realizar el trabajo de levantamiento de requerimientos, desarrollo y entrega de producto en iteraciones, pero de una forma tradicional y monótona, el seguimiento y gestión de desarrollo se llevaría a cabo de una forma controlada, pero la calidad del software tendría sus cuestionamientos ya que no se hubieran tomado las consideraciones de pruebas. Si se hubiese realizado únicamente con Kanban, se hubiese tenido una mejor forma de ver las tareas de todo el proyecto, aquellas por hacer, aquellas en proceso y aquellas ya realizadas, así como el limitar la cantidad de tareas en las que trabaja el equipo simultáneamente, consiguiendo un mayor control de las fases del proyecto, flexibilidad y disminución del tiempo dedicado a tareas poco productivas, pero hubiésemos tenido un resultado de poca calidad, además de reducir la interacción del equipo.

El uso combinado de las características expuestas de estas metodologías referentes a XP, Scrum y Kanban han permitido que este marco de trabajo logre tener aparte de una adecuada gestión de desarrollo de productos de software, el de llevar un control y mejorar la comunicación tanto internamente como externamente, la gestión de calidad en los productos que se van desarrollando se puede apreciar a través de estrictos controles, tales como el uso de pruebas de software, refactorización de código, propiedad colectiva, así como tener una clara visualización del flujo de trabajo.

Referencias

Balza, L. M., Briceño, T., Linares, B., Lobo, R., & Nuñez, C. (2007). Informe sobre XP. Universidad Valle del Momboy. Trujillo, Venezuela.

Bahit, E. (2012). Scrum y Extreme Programming. Argentina, Buenos Aires.

Bermejo, M. (2011). El Kanban. Barcelona, España: UOC.

Casas, S., Reinaga, H. (2009). Aspectos tempranos: Un enfoque basado en tarjetas CRC.Sociedad Colombiana de Computación. https://www.researchgate.net/publication/220136724_Aspecto s_tempranos_Un_enfoque_basado_en_tarjetas_CRC

Canty, D. (2015). Agile for Project Managers. U.S: Taylor & FrancisGroup.

Cifuentes Lozano, A. (2012). Integración de Buenas Prácticas. Recuperado el 24 de noviembre de 2014, de: http://bibliotecadigital.icesi.edu.co/biblioteca_digital/bitstream /10906/68430/5/modelo_integracion_practicas.pdf

Cohn, M. (2006). Planning Poker. Mountain Goat Software, 56-59.

Deemer, P., Benefield, G., Larman, C., & Bas, V. (2009). The Scrum Primer Versión en Español. California, Estados Unidos.

Deemer, P., Benefield, G., Larman, C., & Bas, V. (2009). Información Básica de Scrum the Scrum Primer Version 1.1. Scrum Training Institute. Traducción de Leo Antoli. Agile-Spain. Disponible en: http://www.goodagile.com/scrumprimer/scrumprimer_es.pdf. Consulta: 30 de mayo del 2011

Dimes, T. (2015). Conceptos Básicos de Scrum. (1st Ed.). Babelcube, Inc.

Dooley, J.F. (2017). Software Development, Design and Coding. (2nd Ed.). Springer Science.

Du Bois, B., Demeyer, S., & Verelst, J. (2004, November). Refactoring-improving coupling and cohesion of existing code. In 11th working conference on reverse engineering. IEEE.144-151.

Haugen, N. C. (2006, July). An empirical study of using planning poker for user story estimation. In AGILE 2006 (AGILE'06). IEEE.9.

Janes, A. & Succi, G. (2014). Lean Software Development in Action. Springer Verlag Berlin. doi: 10.1007/978-3-642-00503-9

Joskowicz, J. (2008). Reglas y prácticas en eXtreme Programming. Universidad de Vigo, 22.

Letelier, P., & Penadés, M. C. (2006). Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP). Ciencia y Técnica Administrativa.

Letelier, P., & Penadés, M. C. (13 de marzo de 2012). Repositorio institucional de la Universidad de Las Tunas. Obtenido de http://roa.ult.edu.cu/bitstream/123456789/476/1/TodoAgil.pdf

Kanbanize (2020). Kanbanize. Recuperado el 3 de enero de 2020, de: https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-tablero-kanban/

Koch, A. S. (2005). Agile Software Development. U.S, Norwood: Artech House Inc.

Klipp, P. (2014). Getting Started with Kanban.

Mora, A. (2015). Managing Agile Strategy, Implementation, Organisation and people. Springer International. doi: 10.1007/978-3-319-16262-1.

Maximini, D. (2015). The Scrum Culture Introducing Agile Methods in Organizations. Switzerland: Springer International Publishing.

Moré Martín, A. (2011). MPlu+ a Ágil: el modelo de proceso centrado en el usuario como metodología ágil. Recuperado el 9 de febrero de 2011, de: https://repositori.udl.cat/handle/10459.1/45841.

Ohmyroot. Estándares de codificación. Recuperado el 12 de enero de 2017, de: https://www.ohmyroot.com/buenas-practicas- legibilidad-del-codigo/

Palacio, J. (2007). Flexibilidad con Scrum Principios de diseño e implantación de campos de Scrum. Safe Creative.

Palacio, J. (2019). Scrum Master. España: Lubaris Info 4 Media SL.

Project Management Institute. (2017). Agile Practice Guide. Pennsylvania, U.S. Project Management Institute, Inc.

Robles, G., & Ferrer, J. (2002). Programación eXtrema y Software Libre. Universidad Politécnica de Madrid. España.

Rosas, A.K. (2017). Sistema web para control de inventarios y rendimientos. Repositosio Institucional de Investigación. Universidad Tecnológica del Centro de Veracruz, Recuperado el 4 de abril de 2017, de: http://reini.utcv.edu.mx/handle/123456789/239

Rubin, K. (2013). Essential Scrum. USA: Pearson Education, Inc.

Salamon, A., Maller, P. A., Boggio, A., Mira, N., Perez, S., & Coenda, F. (2014). La integración continua aplicada en el desarrollo de software en el ámbito científico–técnico. In XX Congreso Argentino de Ciencias de la Computación (Buenos Aires, 2014).

Saliu, M. O., & Ruhe, G. (2007). Bi-objective release planning for evolving software systems. In Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (pp. 105-114).

Subra, J.P. & Vannieuwenhuyze, A. (2018). Scrum un Método Ágil Para sus Proyectos. España: ENI.

Schwaber, K. & Sutherland, J. (2017). The Scrum Guide. USA: Org and Scrum Inc.

Stober, T. y Hansmann, U. (2010). Best Practices for Large Software Development Projects. Heidelberg, Germany: Springer-Verlag.

Tesei, F., Cabrera, M., Vaquero, M., & Tedini, D. (2019). Acercando la academia al mundo real: una experiencia de aplicación de Metodologías Agile al proceso de enseñanza-aprendizaje en una asignatura de desarrollo de software. In I Simposio Argentino de Educación en Informática.

Wake, B (2003). Extreme Programming Explored. Massachusetts, U.S: Addison Wesley.