View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Uniandes Episteme (E-Journal - Universidad Regional Autónoma de Los Andes)

METODOLOGÍA HÍBRIDA DE DESARROLLO DE SOFTWARE COMBINANDO XP Y SCRUM

AUTORES: Marcos Klender Carrasco Gonzaga[1]
Willian Javier Ocampo Pazos[2]
Luis Javier Ulloa Meneses[3]
Jon Azcona Esteban[4]

DIRECCIÓN PARA CORRESPONDENCIA: mkcarrascog@pucesd.edu.ec

Fecha de recepción: 11-04-2019

Fecha de aceptación: 27-05-2019

RESUMEN

El presente artículo tiene como objetivo describir una forma adecuada de combinar la metodología de desarrollo ágil XP y el marco de trabajo Scrum, explicando cómo se complementan entre sí. Del lado de XP, se destacan las prácticas, valores y el ciclo de vida que esta metodología propone, misma que se compone de seis fases: Exploración, Planeación, Diseño, Codificación, Pruebas y Muerte del Proyecto. En lo que respecta a Scrum, se destacan los eventos y artefactos que posee este marco de trabajo para cubrir las necesidades del producto. La combinación de XP y Scrum supuso una gran ayuda en el proceso de desarrollo de software, evitando la documentación exhaustiva y haciendo del cliente un miembro más del equipo.

PALABRAS CLAVE: Desarrollo de Software; Metodologías; Marco de Trabajo; XP; Scrum.

HYBRID METHODOLOGY OF SOFTWARE DEVELOPMENT COMBINING XP AND SCRUM

ABSTRACT

The purpose of this article is to describe an adequate way to combine the Agile XP development methodology and the Scrum framework, explaining how they complement each other. On the XP side, we highlight the practices, values and life cycle that this methodology proposes, which consist of six phases: Exploration, Planning, Design, Coding, Testing and Death of the Project. With regard to Scrum, we highlight the events and artifacts that this framework has to cover the needs of the product. The combination of XP and Scrum was a great help in the software

[1] Ingeniero de Sistemas y Computación, Desarrollador Independiente (Freelancer), Pontificia Universidad Católica del Ecuador sede Santo Domingo. Santo Domingo de los Tsáchilas, Ecuador. E-mail: mkcarrascog@pucesd.edu.ec

[2] Ingeniero en Sistemas e Informática, Magíster en Gerencia Educativa, Maestrante en Dirección e Ingeniería de Sitios Web Rama Investigación, Profesor tiempo completo de la Escuela de Sistemas de la Pontificia Universidad Católica del Ecuador sede Santo Domingo, Ecuador. E-mail: opwj@pucesd.edu.ec

[3] Ingeniero en Sistemas e Informática, Magíster en Informática Empresarial, Profesor tiempo completo de la Escuela de Sistemas de la Pontificia Universidad Católica del Ecuador sede Santo Domingo, Ecuador. E-mail: umlj@pucesd.edu.ec

[4] Ingeniero en Automática y Electrónica Industrial, Máster en Integración de las Energías Renovables en el Sistema Eléctrico, Profesor tiempo completo de la Escuela de Sistemas de la Pontificia Universidad Católica del Ecuador sede Santo Domingo, Ecuador. E-mail: aej@pucesd.edu.ec

© Centro de Investigación y Desarrollo. Universidad Regional Autónoma de Los Andes - Extensión Santo Domingo. Ecuador.

development process, avoiding exhaustive documentation and making the client one more member of the team.

KEYWORDS: Software Development; Methodologies; Framework; XP; Scrum.

INTRODUCCIÓN

La ingeniería de software se interesa por todos los aspectos de la producción de software, empezando desde las etapas más tempranas del sistema hasta el mantenimiento que se le da después de implementar el software. Además, no sólo se interesa en los procesos técnicos del desarrollo de software, también lo hace con la administración del proyecto y el desarrollo de herramientas, aplicando teorías y métodos donde sea adecuado para obtener resultados de la calidad requerida dentro de una fecha establecida (Sommerville, 2011, p. 7).

Según Vlaanderen, Jansen, Brinkkemper & Jaspers (2011), en los últimos años se ha introducido los principios ágiles como una de las mayores innovaciones en las metodologías de software. Uno de los puntos fuertes de la metodología ágil es que el trabajo se vuelve dinámico, aceptando los cambios que se puedan dar a lo largo del desarrollo, colaborando con los clientes y eligiendo al software por encima de la documentación exhaustiva. Dicha metodología da paso a los equipos ágiles, cuyo potencial se ve reflejado en la toma de decisiones y en el apoyo que se brindan entre los integrantes del equipo, a diferencia de lo complejo que llegarían a ser estos procedimientos de forma individual (Drury, Conboy & Power, 2012).

Lo anterior se logra gracias a la aplicación de procesos ágiles, los cuales son métodos que le permiten al equipo de desarrollo enfocarse en el software en lugar del diseño y la documentación innecesaria. Un método ágil permite que el desarrollo de un producto sea incremental, es decir, en un inicio crear un software sencillo que vaya adoptando nuevas características a medida que va progresando su desarrollo (Pressman, 2010).

A diferencia de otros métodos, aquí no se crean partes del producto por separado para al final unir todos los módulos desarrollados, sino que se desarrollarán características funcionales contemplando el producto desde un inicio. Además, se debe tomar en cuenta que para que un método pueda considerarse ágil debe poder adaptarse a los cambios que puedan surgir en el transcurso del desarrollo de un producto (Álvarez, de las Heras del Dedo, & Lasa, 2012, p. 34).

En la ingeniería de software existen decenas de metodologías, modelos y herramientas, muchas de ellas ya obsoletas. En su momento, cada una de ellas tuvo gran acogida y respaldo, pero poco a poco fueron reemplazadas por nuevas metodologías que se enfocaban en lo moderno y adaptativo. El decir que estas metodologías pueden combinarse con otras es indiscutible y en la presente investigación se detallará cómo Scrum y XP pueden complementarse mutuamente.

DESARROLLO

Programación Extrema (XP)

La Programación Extrema es una de las metodologías ágiles más utilizadas en el desarrollo de software cuyos requisitos son poco complejos o cambiantes. Se aplica mayormente en equipos de desarrollo pequeños o medianos, llevando una clara orientación tanto con los desarrolladores como con los clientes o usuarios finales. En esta metodología los programadores trabajan en pares, desarrollando pruebas para cada función que quieran implementar (conocidas como

tareas). Todas las pruebas que se realicen deben funcionar correctamente antes de integrar una nueva característica en el sistema (Álvarez et al., 2012, p. 49).

A continuación, se presentan las principales características de XP que se complementan adecuadamente al marco de trabajo Scrum para el desarrollo de software ágil.

Tabla 1. Características aplicables de XP

Programación Extrema (XP)	
Estándares de Codificación	Aquí se ha especificado un estilo y formato para el código fuente. Esta práctica mantiene el código consistente, facilitando la lectura, interpretación y refactorización que se pueda dar en un futuro.
Diseño Simple	Se ha hecho especial énfasis en mantener el código lo más simple posible, priorizando las historias de usuario planeadas y evitando dar importancia a las funcionalidades que se implementen a futuro.
Refactorización	Se han realizado cambios pertinentes en el código para mantenerlo limpio y legible, con el objetivo de que sea más fácil añadir características sin alterar la funcionalidad del sistema.
Integración Continua	Aquí se establece que cada tarea realizada se ha integrado al sistema, haciendo uso de las pruebas unitarias a fin de corroborar que el agregado no perjudica las funcionalidades existentes.
Pruebas	Se han realizado tres tipos de pruebas: Unitarias (que constituyen el 70% del total de pruebas realizadas), las que se realizaron con mayor frecuencia; Integración (20%), ejecutadas al incluir nuevas funcionalidades; Aceptación (10%), realizadas por el usuario. Este patrón de pruebas está basado en la Pirámide de Cohn (Cohn, 2009).

Nota: Adaptado de "Scrum y XP desde las Trincheras" por H. Kniberg, 2015.

Proceso XP

La programación extrema consta de seis actividades estructurales: Exploración, Planeación, Diseño, Codificación, Pruebas y Muerte del Proyecto. A continuación, se explicarán de manera resumida las cuatro principales fases de proceso XP.

La *Planeación* comienza recabando la información necesaria para empezar con el desarrollo del sistema, creando las historias de usuarios iniciales y el plan de iteración. El *Diseño* guía la implementación de una historia de usuario; aquí se utilizan las tarjetas CRC y se hace uso del principio MS (mantenlo sencillo) para todo en lo que se esté trabajando. Luego, en la etapa de *Codificación*, se desarrollan pruebas unitarias y se estimula la programación en parejas; a medida que las funcionalidades se van desarrollando se aplican estas pruebas, con el fin de retroalimentar el avance del producto. Finalmente, en la etapa de *Pruebas* se realizan las pruebas de aceptación, de manera que se verifique el funcionamiento del incremento de software que se ha realizado en la iteración (Álvarez et al., 2012, pp. 62-65).

Scrum

De acuerdo con Scrum Manager (2016), Scrum técnico es un marco de trabajo que comprende varios procesos y técnicas orientadas a la gestión del desarrollo de productos complejos. Busca generar resultados de calidad en iteraciones cortas (generalmente de 2 a 4 semanas), en las cuales el equipo de desarrollo hace uso de los eventos, artefactos y reglas asociadas.

Cabe destacar que Scrum emplea un enfoque incremental que se adapta a los cambios que afecten al sistema, además de basarse en la experiencia para predecir y controlar riegos en el desarrollo (Schwaber & Sutherland, 2017).

A pesar de que Scrum surgió como un modelo para la gestión del trabajo de productos complejos, también se aplica en proyectos de software donde los requerimientos varían a medida que se desarrolla el sistema, donde la innovación y la adaptabilidad son parte fundamental del desarrollo y cuando se necesitan pequeñas liberaciones funcionales. Además, aquí se hace especial énfasis en el alineamiento entre el cliente y el equipo de desarrollo, de modo que el sistema se enriquezca de las aportaciones de todos los involucrados (Brito, Sosa, & Héctor, 2015, pp. 40-42).

A continuación, se presentan las principales características de Scrum que complementan adecuadamente a la metodología XP para el desarrollo de software ágil.

Tabla 2. Características aplicables de Scrum

Scrum	
Product Backlog	Aquí se han enlistado todas las historias de usuario que el Product Owner ha definido, organizándolas de acuerdo a la prioridad, la estimación y el riesgo de desarrollo que se ha estimado.
Sprint Backlog	Aquí, en cambio, se han enlistado las historias de usuario que se realizarán a lo largo del sprint. Este artefacto también hace uso del gráfico Burndown Chart para medir el trabajo pendiente.
Daily Scrum	Esta reunión es indispensable para verificar las tareas que han realizado el día anterior, visualizar las tareas que se desarrollarán a lo largo del día e identificar algún obstáculo que se haya presentado.
Sprint Review	Esta reunión se realiza al final de cada sprint con el objetivo de verificar las funcionalidades que se han realizado (Done), las que no y el por qué, de manera que se corrijan estas falencias para el próximo sprint.
Sprint Retrospective	Esta reunión se produce a fin de analizar detenidamente el trabajo realizado a lo largo del sprint. De esa manera se han detallado los puntos fuertes y débiles del proceso que se lleva a cabo.

Nota: Adaptado de "Scrum y XP desde las Trincheras" por H. Kniberg, 2015.

Equipo Scrum

El equipo Scrum se compone de tres roles: Product Owner, Development Team y Scrum Master. El *Product Owner* es una única persona responsable de optimizar el trabajo que realizan los desarrolladores y de gestionar el Product Backlog (este es un artefacto que se lo definirá más adelante). Se lo puede entender como el cliente que solicita el producto, pero también puede ser un intermediario entre los que requieran el producto y el equipo de desarrollo. El *Development Team* son todos aquellos encargados de realizar el producto que necesita el cliente. El equipo no tiene jerarquía y su tamaño óptimo es de tres a nueve integrantes. Por último, el *Scrum Master* es el encargado de asegurar que todo el equipo entienda y adopte la teoría, prácticas y reglas de Scrum a la perfección (Navarro, Fernández, & Morales, 2013).

Resultados

La combinación de las características descritas anteriormente mejora todo el proceso de desarrollo de software, empezando por el levantamiento de la información. Las historias de usuario facilitan la tarea de recopilar las funcionalidades del cliente y se consideran una excelente alternativa a los Sprint Backlog Items convencionales.

Haciendo referencia a XP, se considera que el ciclo de vida que propone esta metodología es el adecuado para el producto, en vista de que se enfoca totalmente al desarrollo de software. Tomando en consideración que el Development Team sólo lo conforma el autor, se deduce que algunas prácticas XP no pueden ser aplicadas (como la programación en parejas, por ejemplo), pero las demás prácticas mencionadas anteriormente sí, debido a que brindan grandes beneficios para el desarrollo del producto y la calidad de vida del programador.

Un código limpio, simple y legible facilita enormemente los procesos de refactorización e integración del software, además de que lo hace escalable a futuro. De igual manera, las pruebas realizadas a lo largo del desarrollo son fundamentales para la entrega de un producto funcional que cumpla las expectativas del cliente; por otra parte, dicho desarrollo fue dividido en sprints, por lo cual se puede observar lo bien que Scrum puede complementar a XP.

En lo que respecta a Scrum los artefactos como el Product Backlog organiza las historias de usuario de acuerdo a la prioridad que el cliente les haya otorgado; de la misma forma, el Sprint Backlog organiza las tareas de ingeniería que componen cada historia de usuario y lleva un control sobre el desarrollo de cada una de ellas. Los eventos también resultan indispensables en el desarrollo del producto, puesto que permiten verificar diariamente el trabajo realizado y las tareas pendientes, analizar los problemas que han surgido, las posibles soluciones que se pueden aplicar y las mejoras que se pueden poner en práctica posteriormente.

Ciclo de Vida XP

Para todo el proceso de desarrollo, XP propone seis fases: Exploración, Planeación, Diseño, Codificación, Pruebas y Muerte del Proyecto. De igual manera, Scrum propone el uso de diferentes artefactos y eventos propios del marco de trabajo, los mismos que serán muy útiles a lo largo del ciclo de vida del software.

Es necesario destacar que el trabajo realizado para el desarrollo del producto se ha dividido en sprints (Gestión evolutiva con incremento iterativo), lo cual facilita la organización global de cada uno de ellos haciendo uso de las herramientas que brinda Scrum.

Fase I: Exploración

Para empezar, es necesario definir los roles de las personas involucradas en el proyecto, como lo pueden ser: Product Owner, Development Team y Scrum Master.

En esta primera fase se ha buscado realizar una reunión con el Product Owner y con todos los interesados del negocio, el cual realiza la función de intermediario entre los interesados y el equipo de desarrollo para recolectar las funcionalidades que la empresa necesita, con la finalidad de que se puedan plantear las historias de usuario que son de interés para la entrega del producto. Estas historias son los ítems del Product Backlog.

Fase II: Planeación

A partir de esta fase se da inicio con el primer sprint. Para empezar, se eligen las historias de usuario de acuerdo a la prioridad que se tienen en el Product Backlog, cuyos puntos de estimación totales no superen la velocidad de desarrollo del equipo. Una vez que se han establecido las prioridades de las historias de usuario, el Development Team procede a estimar la complejidad de cada una de ellas haciendo uso de los puntos de historia, métrica utilizada en las metodologías de desarrollo ágil. Esto último se logra asignando un valor estimado a la

complejidad (haciendo uso de la serie Fibonacci) de acuerdo a la experiencia de los desarrolladores.

Por consiguiente, se crea el Sprint Backlog en base a las historias de usuario elegidas, dividiéndolas en tareas de ingeniería para llevar un mejor control del trabajo pendiente, la duración de cada sprint que será de cuatro semanas. Se recomienda que en esta fase se apoye en la técnica mockps, para realizar una fertilización cruzada, escuchar las necesidades del Product Owner (voz y mente del cliente) y se define los escenarios de pruebas.

Gestión de Tareas

Con el objetivo de llevar un control preciso de las historias de usuario y sus respectivas tareas de ingeniería, se ha empleado un software para la gestión de tareas llamado Trello. Este software hace uso del sistema Kanban (sistema de tarjetas) en una interfaz web que ha facilitado la gestión del desarrollo del proyecto de acuerdo a la organización que propone Scrum, donde se controla el trabajo pendiente y se organiza el trabajo terminado en las diferentes columnas: en espera, en desarrollo, por validar y completo. A manera de ejemplo, se presenta el enlace al tablero Trello que se ha realizado con anterioridad: https://trello.com/b/lZ7ApRPF.

Gestión de Versiones

Para facilitar los procesos de respaldo del sistema y gestionar las versiones que se vayan desarrollando, se optó por utilizar Github, una excelente plataforma enfocada al desarrollo colaborativo de software. Github utiliza Git, un sistema de control de versiones que permite alojar repositorios en la web, permitiendo crear una copia local del código y respaldarla para su posterior revisión, uso y gestión; además, permite volver a versiones anteriores del sistema en caso de que se haya presentado algún error.

Fase III: Diseño

La fase de diseño se enfoca en el uso de las tarjetas CRC (Clase-Responsabilidad-Colaborador). Estas tarjetas ayudan a identificar y organizar fácilmente las clases más relevantes para los requerimientos de un sistema; además, se dividen en tres secciones: en la parte superior se encuentra el nombre de la clase, en la parte izquierda se enlistan todas las acciones que la clase pueda realizar (responsabilidades) y en la parte derecha, otras clases que interactúen con la actual (colaboradores), ayudándola a llevar a cabo sus responsabilidades.

Fase IV: Codificación

Aquí es donde se desarrollan las funcionalidades del sistema, escribiendo el código fuente necesario para cumplir con lo detallado en las tareas de ingeniería. Cabe mencionar que diariamente se ha realizado el evento Daily Scrum, reunión que ayudó a verificar el trabajo realizado el día anterior, visualizar las tareas que se desarrollarán a lo largo del día e identificar los riesgos que se han presentado.

Fase V: Pruebas

Es necesario aclarar que, a lo largo de la fase anterior, se realizaron pruebas unitarias automatizadas con el objetivo de verificar que el desarrollo continuo no afecta a otras funcionalidades. Retomando esta última fase del sprint, se utilizó el evento Sprint Review, reunión en compañía del Product Owner donde se realizó la demostración del producto mínimo viable (incremento), explicando las funcionalidades desarrolladas; además, esta reunión se apoyó

en las pruebas de aceptación, las cuales sirven como constancia de que el incremento fue desarrollado con éxito.

Una vez terminado el desarrollo del sprint, como resultado del control diario de las tareas de ingeniería (Daily Scrum) se obtendrá el gráfico de avance, también conocido como Burndown Chart, el cual permite apreciar las tareas pendientes. Finalmente, también se realizó el evento Sprint Retrospective, reunión realizada luego de la revisión del sprint, cuyo objetivo es el de analizar detenidamente el trabajo realizado a lo largo del sprint para identificar las diferentes maneras de cómo mejorar la forma de trabajo.

A partir de esta fase se da por concluido el sprint correspondiente. Es necesario volver a la fase de Planeación (Fase II) para continuar con el siguiente sprint y así sucesivamente. Una vez que todos los sprints hayan sido culminados con éxito, se procede a la siguiente fase.

Fase VI: Muerte del Proyecto

Una vez que el Product Backlog no tiene más historias de usuario y todas las funcionalidades requeridas fueron desarrolladas satisfactoriamente, se da por terminado el ciclo de vida de desarrollo del software. En esta última fase se realiza la documentación final que corrobora la entrega y aceptación del sistema por parte del Product Owner, como por ejemplo una Acta de Entrega-Recepción del Proyecto.

CONCLUSIONES

Gracias a las historias de usuarios utilizadas como instrumento de recolección de datos (funcionalidades) se logró recabar toda la información necesaria para el desarrollo del sistema; y junto con la frecuente comunicación con el Product Owner, se generó la retroalimentación necesaria para solventar cualquier error que se presentó a lo largo de los sprints, dando como resultado un producto confiable, robusto y funcional.

La combinación de la metodología XP y el marco de trabajo Scrum fueron de gran ayuda en el proceso de desarrollo del software. Los procesos y el ciclo de vida ideal que propone XP junto con los eventos y artefactos Scrum aportaron vital importancia para que el desarrollo sea versátil y limpio, de manera que permita realizar cualquier cambio en el producto que surja a lo largo de los sprints, sin necesidad de comprometer las funcionalidades ya desarrolladas.

REFERENCIAS BIBLIOGRÁFICAS

Álvarez, A., de las Heras del Dedo, R., & Lasa, C. (2012). *Métodos Ágiles y Scrum.* Madrid: Anaya Multimedia.

Brito, K., Sosa, D., & Héctor, K. (2015). *Selección de Metodologías de Desarrollo para Aplicaciones Web.* San Bernardino: Académica Española.

Cohn, M. (2009). *The Forgotten Layer of the Test Automation Pyramid*. Mountain Goat Software. Obtenido de: https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid

Drury, M., Conboy, K. & Power, K. (2012). *Obstacles to decision making in agile software development teams.* Journal of Systems and Software, 85(6), 1239-1254. doi: 10.1016/j.jss.2012.01.058

Kniberg, H. (2015). *Scrum y XP desde las Trincheras.* Estados Unidos: C4Media Inc.

Navarro, A., Fernández, J., & Morales, J. (2013). Revisión de metodologías ágiles para el desarrollo de software. *PROSPECTIVA, 11*(2), 30-39.

Pressman, R. (2010). *Ingeniería de Software: Un Enfoque Práctico.* México: McGraw-Hill.

Schwaber, K. & Sutherland, J. (Noviembre de 2017). *The Scrum Guide, The Definitive Guide to Scrum: The Rules of the Game.* Obtenido de http://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100

Scrum Manager. (2016). *Scrum Manager BoK, TRONCAL I: Scrum Master.* Obtenido de http://scrummanager.net/files/scrum_manager.pdf

Sommerville, I. (2011). *Ingeniería de Software.* México: Pearson Educación.

Vlaanderen, K., Jansen, S., Brinkkemper, S. & Jaspers, E. (2011). *The agile requirements refinery: Applying SCRUM principles to software product management*. Information and Software Technology, 53(1), 58-70. doi: 10.1016/j.infsof.2010.08.004