ISSN 1870-4069

Mixing Scrum-PSP: Combinación de Scrum y PSP para mejorar la calidad del proceso de software

Mauricio Leonardo Urbina Delgadillo, María Antonieta Abud Figueroa, Gustavo Peláez Camarena, Giner Alor Hernández, Alma Ivonne Sánchez García

División de Estudios de Posgrado e Investigación, Instituto Tecnológico de Orizaba, Veracruz, México

mauricio.urbina.isc@gmail.com, mabud@ito-depi.edu.mx, sgpelaez@yahoo.com.mx, galor@itorizaba.edu.mx, alivsaga@hotmail.com

Abstract. El desarrollo de software en un entorno real es muy agresivo con respecto a los tiempos de entrega, presupuesto y el contrato con el cliente. La combinación de métodos ágiles y procesos disciplinados es una opción prometedora que apoya al desarrollo de software a mejorar la calidad. En este trabajo se presenta un modelo de integración de procesos combinando Scrum y PSP (Personal Software Process) para mejorar el proceso de software, teniendo en cuenta las características primordiales de cada marco de trabajo. El resultado de esta investigación es un intento para adoptar la agilidad y disciplina de los procesos en diferentes ambientes de trabajo.

Keywords: Calidad de software, gestión del proceso de software, mejora de proceso de software, proceso personal de software, PSP, scrum.

Mixing Scrum-PSP: Combining Scrum and PSP to Improve Software Quality Process

Abstract. Software development into the enterprise environment is characterize by aggressive delivery times, low budget and customer contract. Combination of agile methods and process disciplines is a suitable option for the software development process to improve the quality. In this paper we show a proposal for integrate a process model between Scrum and Personal Software Process to improve the quality of the development process, considering the main characteristics of each approach. The result of this research is an attempt to adopt the agility and discipline in different work environments.

Keywords: software quality, software process management, software process improvement, personal software process, PSP, Scrum.

1. Introducción

El desarrollo de software en un entorno de proyecto real se caracteriza por ser rápido, cambiante y complejo. Debido a que los problemas requieren soluciones especializadas, los ingenieros de software se enfrentan a calendarios agresivos y realistas para generar un producto de calidad sobresaliente. Otro factor que se une es la impaciencia por terminar en el plazo acordado, no exceder el presupuesto y cumplir con los requisitos del cliente merman la calidad del producto final.

Gracias a los métodos ágiles como SCRUM, XP, Dynamic System Development Method (DSDM), Crystal, Lean Development (LD), los equipos de desarrollo de software realizan entregas en poco tiempo y los proyectos se adaptan a las condiciones cambiantes del cliente. Por otra parte, los modelos de calidad para la administración del proceso de desarrollo como CMMI-DEV, SPICE, Six Sigma y PSP/TSP permiten tener una buena gestión del proceso utilizando guías para supervisar que los desarrolladores sigan el flujo establecido por la empresa y ayudan a mejorar calidad del producto final.

Los métodos agiles y los procesos disciplinados representan casos de extremo a extremo en sus paradigmas, y siempre habrá pequeñas discordias entre ambos por parte de los desarrolladores de software [1]. Sin embargo, los dos enfoques tienen características para complementarse uno al otro. Las grandes compañías y los grandes proyectos de software utilizan ambos enfoques en la medida de sus objetivos y su entorno.

Uno de los métodos agiles más populares es Scrum, considerado como un marco de trabajo [2] [3] formado por un conjunto de prácticas y reglas, el desarrollo del software se realiza de manera iterativa e incremental en flujos de trabajo rápidos donde se construye un incremento funcional del sistema. Del lado de los procesos disciplinados se encuentra PSP^{SM}, conocido un marco de trabajo estructurado de formularios, normas, y procedimientos para el desarrollo de software [4], su propósito es ayudar a mejora de las habilidades de ingeniería de software a los desarrolladores.

Esta investigación está enfocada en la relación entre Scrum y PSP para mejorar la calidad del software. Por esta razón se plantea la opción de combinar Scrum y PSP para mejorar el proceso para desarrollo de software en diferentes entornos. Este documento se estructura de la siguiente manera: en la Sección 2 se presentan a Scrum y PSP. En la Sección 3 se mencionan trabajos relacionados con la investigación. La Sección 4 presenta la integración entre Scrum y PSP. El diseño del caso de estudio se encuentra en la Sección 5. Las conclusiones y las líneas de trabajo futuro se describen en la Sección 6.

2. Scrum y PSP

2.1. Scrum

Jeff Sutherland aplicó los principios al desarrollo de software en 1993 en Easel Corporation y no fue hasta 1996 junto con Ken Schwaber que se presentaron como válidos para gestionar el desarrollo de software [3]. Es definido como un marco de trabajo por el cual las personas pueden trabajar con problemas complejos adaptativos, al mismo tiempo se entregan productos de gran valor posible. Caracterizado por ser

ligero, fácil, y difícil de dominar; está más orientado a las personas que a los procesos porque la gestión no se basa en el seguimiento de un plan sino en la adaptación continua a las circunstancias de un proyecto.

El ciclo de vida de Scrum se compone de tres fases *Pre-juego*, *Juego* y *Post-juego* (Figura 1). Scrum incluye tres características consideradas críticas para la implementación del control de procesos empíricos [5]:

- ***Transparencia:*** los procesos significativos deben ser visibles para aquellos que son responsables del resultado. Se requiere que dichos aspectos sean definidos por un estándar común, de tal modo que los involucrados compartan un conocimiento general de lo que está viendo.
- ***Inspección:*** Los involucrados tienen que inspeccionar con frecuencia los artefactos generados y el progreso hacia el objetivo del "sprint" para detectar variaciones indeseables. Estas inspecciones no tienen que ser muy frecuentes para que no interrumpa el ciclo de trabajo.
- ***Adaptación:*** Si la inspección determina que uno o más aspectos de un proceso se alejan de los límites aceptables, y que el producto resultante no será aceptado, el proceso o el material que se está siendo trabajando tiene que ser ajustado. El ajuste debe realizarse para minimizar desviaciones mayores.

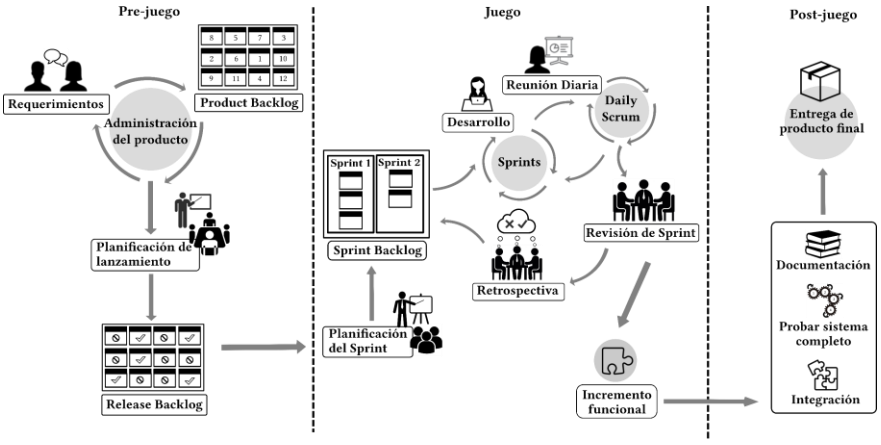

Fig. 1. Ciclo de vida de Scrum.

2.2. Personal Software Process (PSP)

PSP en un marco de trabajo para mejorar los procesos de construcción de software a un nivel personal, su estrategia permite mejorar el rendimiento de los programa-dores utilizando prácticas sólidas para monitorizar y esforzarse en mejorar el desempeño, y la calidad de los productos. PSP se basa en los siguientes principios de planificación y calidad [4]:

- Cada desarrollador es diferente, los desarrolladores tienen que planear su trabajo con base a su información personal.

- Para una mejorar continua del desempeño, los desarrolladores definen y miden correctamente sus procesos.
- Para producir productos de calidad, los desarrolladores deben sentirse responsables de la calidad de su trabajo.
- El costo es menor si se encuentran y reparan defectos en fases tempranas que en fases próximas.
- Es más eficiente prevenir los defectos que buscarlos y repararlos.
- El camino correcto el siempre el más rápido y barato de trabajar.

PSP define siete niveles de madurez (Figura 2), los niveles PSP0 y PSP1 son críticos para aprender la disciplina del proceso, considerados punto de partida para la secuencia del proceso. Cada nivel de PSP se dividen en fases o procesos, descrita por medio de un *Scripts* el cual no solo especifica los pasos a seguir también los criterios de entrada y salida. Humphrey se centra en tres fases: *diseño*, *código* y *pruebas* pues son las actividades que más tiempo consumen en el desarrollo del software.

3. Trabajos Relacionados

En este apartado se presentan investigaciones de varios autores realizadas con un objetivo similar entre ellas, la combinación de métodos agiles y métodos disciplinados para mejorar la ingeniería de software en factores como la gestión de procesos y administración de la calidad.

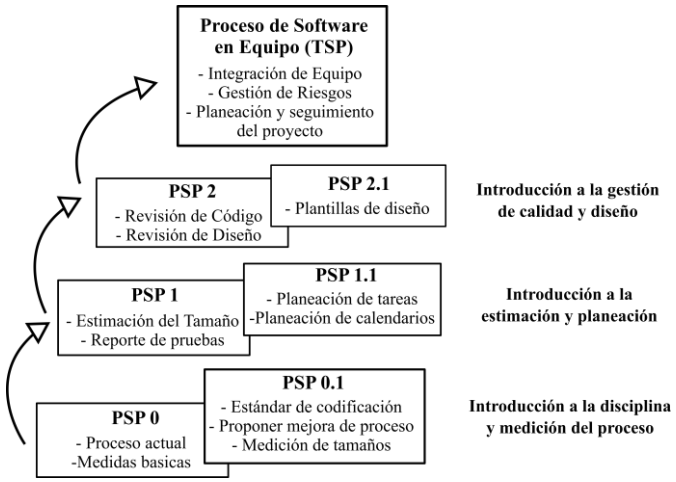

Fig. 2. Niveles de Madurez de PSP (Fuente: PSP: A Self-Improvment Process for Software Engi-neers, p. 8).

A continuación se presenta un cuadro comparativo (Tabla 1) que permite una rápida y clara perspectiva de trabajo realizado en las investigaciones, la primera columna muestra los métodos utilizados por los autores, la segunda describe el análisis utilizado en la investigación, la tercera muestra brevemente la combinación resultante.

Tabla 1. Análisis comparativo entre trabajos anteriores relacionados con la modificación entre métodos ágiles y métodos disciplinados.

Métodos utilizados	Diseño metodológico	Combinación resultante
Boehm,2002 [6]		
Agiles: XP, Adaptive Software Development, Crystal, DSDM. *Disciplinados:* basados en riesgos, basados en planes, CMM, CMMI.	Comparación entre 7 áreas clave para cada enfoque (*Desarrolladores, Clientes, Requerimientos, Arquitectura, Ajustes, Tamaño* y *Objetivo principal*) en donde cada método tiene ventajas en determinados proyectos.	Por sugerencia del autor, realizar análisis de riesgos basado en las características del proyecto utilizando las 7 áreas descritas para determinar un balance entre los enfoques.
Paulk, 2002[7]		
Agiles: Principios del manifestó ágil y XP. *Disciplinados:* CMM y SW-CMM.	Comparación en contra entre las características: *Velocidad / Disciplina, Individuos / Procesos, Desarrollo / Documentación, Respuesta a cambios / Planeación, Colaboración de Clientes / Contratos*.	Por sugerencia del autor, recolectar información sobre el tipo de proyecto, generar un análisis considerando la naturaleza del proyecto para la integración de prácticas de ambos enfoques.
Sutherland, 2007[8]		
Ágil: Scrum y Lean. *Disciplinado:* CMMI nivel 2 y 3.	A través de 12 Practicas Genéricas asociadas con CMMI se establece la disciplina dentro de los proyecto de una empresa. Scrum ayuda a resolver problemas de CMMI.	Combinación entre CMMI, Scrum y Lean. Ofrece disciplina y agilidad para la institucionalización de procesos dentro de una empresa.
Jerzy Nawrocki, 2001 [9]		
Ágil: XP. *Disciplinados:* CMMI.	Estableciendo un análisis para relacionar diferentes prácticas de XP con Prácticas Especificas de CMMI nivel 2.	eXtrem Programming Maturity Model (XPMM) de cuatro niveles: *No cumple con nada, Inicial, Avanzado, Maduro*. Un proceso incremental definido por un conjunto de prácticas obligatorias de XP para cada nivel del proceso.
Yani Dzhurov, 2009 [10]		
Agiles: XP. *Disciplinados:* PSP.	Modificación de PSP para aligerar y hacer más fácil el proceso de desarrollo manteniendo sus principios de básicos. Con 6 prácticas de PSP y 6 prácticas de XP se estructuró una metodología que apunta hacia una mejor planeación y control de la calidad del desarrollo de software.	Personal eXtrem Programming (PXP) es un proceso iterativo y comprende un par de iteraciones y ciclos controlados. Conformado por siete fases: *Requerimientos, Planificación, Iteración inicial, Diseño, Implementación, Pruebas de Sistema, Retrospectiva*.

Métodos utilizados	Diseño metodológico	Combinación resultante
Brown, 2014 [11]		
Agiles: Scrum. *Disciplinados:* PSP y SEMAT.	Basado en un análisis de las adaptaciones Scrum-PSP y PXP se propuso una adaptación que reutilice las prácticas de PSP en cualquier método de desarrollo de software. Se utilizó la notación de SEMAT para expresar en un lenguaje común las prácticas de PSP.	Integración con el núcleo SEMAT, muestra a PSP como una alpha "*way of working*". Los 7 niveles de PSP son representados como estados de un sub-alpha "*PSP Compliance*". Igualmente se adapta Scrum utilizando el Backlog, el incremento y el Sprint como alphas dentro de SEMAT.

4. Mixing Scrum-PSP: agilidad y disciplina trabajando juntos

En esta sección se presenta la integración de los dos métodos, Scrum y Personal Process Software (PSP), como primer punto se muestra un análisis de la compatibilidad que existe entre ambos enfoques basado en sus características, puntos fuertes y debilidades; por último se describe la integración resultante.

4.1. Análisis de compatibilidad

Este análisis tiene como propósito generar una comparación para determinar las características de integración y las actividades que sirven para mejorar el proceso de desarrollo del software (Tabla 2).

Una brecha encontrada en Scrum es la carencia de pautas para el proceso de desarrollo que orienten al desarrollador en la construcción del incremento, aquí es donde PSP brinda una serie de prácticas sencillas para la construcción del incremento. Además, gracias a la recolección de datos, por parte de PSP, genera registros donde la información es utilizada para predecir un mejor tiempo de trabajo en las actividades.

La esencia de Scrum son: la agilidad, adaptación a cambios, la estimación del tiempo de trabajo y el proceso iterativo e incremental. Estas características son consideradas parte fundamental del esqueleto de la integración. Por otro lado, para tener una base sólida existen 5 expectativas de PSP, mencionadas por Humphrey [3], que deben cumplirse en cualquier implementación: La definición las métricas y/o características de calidad que se aplicarán al proyecto y su producto, Cumplir con la recolección de los datos de todas las actividades realizadas en los proyectos para generar la base de conocimientos que permitirá hacer las estimaciones a futuro, Obligatorio que todos los integrantes del equipo dispongan de los recursos con libertad y facilidad que les ayuden a realizar sus actividades, Cada integrante del equipo tiene que seguir el flujo del proceso con disciplina, Realizar la medición del producto final y por ende que cumpla con la meta establecida.

Tabla 2. Análisis comparativo entre factores de Scrum y PSP.

Factores	Scrum	PSP
Enfoque	Agilidad en desarrollo. Adaptación a cambios e Inspecciones del trabajo. Producción rápida.	Mejorar las habilidades personales del desarrollador. Estimación basada en datos históricos. Predictibilidad.
Conocimiento	Empírico, Tácito.	Teórico basado en registros.
Prácticas	Gestión del proceso centrado en los requerimientos del cliente. Priorización de los requerimientos y estimación del tiempo. Descomposición de tareas.	Establece un flujo de trabajo para el ingeniero de manera personal. Define guías (scripts) para la administración del proceso.
Ciclo de vida	Iterativo e incremental. Define tres fases: Pre-juego, Juego y Post-juego.	Iterativo y escalonado por niveles. Define tres principales procesos: Planificación, Desarrollo y Postmortem.
Ambiente	Cambiante, inquieto, rápido, enfocado al proyectó.	Estable, pocas modificaciones, enfocado a la disciplina.
Cultura	Trabajo y colaboración en equipo. Permite una mejora continua para todos los integrantes del equipo.	Establece la disciplina y el respeto al proceso de trabajo. Establece una mejora continua de las habilidades personales del desarrollador.

4.2. Mixing Scrum-PSP (MSP)

La integración de Scrum con PSP proporciona pautas para los integrantes del equipo de desarrollo en mejorar la administración del proceso con simples prácticas. *MSP* está formado por dos capas de procesos: el *Ciclo de Vida MSP* y La *Iteración de Desarrollo*.

Ciclo de Vida MSP. Definido por las tres fases *Preparación*, *Desarrollo* y *Entrega* (Figura 3), las fases de planeación y entrega son compaginadas con las fases de *Pre-juego* y *Post-juego* de Scrum, se cambiaron los nombres para una mejor identificación por parte de los involucrados.

La fase de *Preparación:* Abarca la definición del proyecto, la obtención de las necesidades del cliente que posteriormente se convertirán en requisitos del sistema y la creación del *Product Backlog*. La planificación estratégica de recursos es conforme al *Product Backlog* para determinar un calendario de entregas y estimación de tiempos para el proyecto, también se realiza un plan de desarrollo y análisis de riesgos. Conforme los requisitos son priorizados por el equipo de desarrollo se seleccionan los que componen el *Backlog de Liberación*. Las estimaciones estarán apoyadas por el método estadístico *PROBE* de PSP, el cual tiene buena precisión para calcular el costo de las actividades en esfuerzo y tiempo, según los informes del *Software Engineering Institute* (SEI).

La fase de *Desarrollo*: El equipo de desarrollo elige cuales son las actividades del *Backlog de Liberación* que entrarán el *Sprint Backlog* y el número de sprint utilizados para desarrollar todos los requerimientos. Se integran las prácticas de PSP basadas en las tres principales fases: *Planeación*, *Desarrollo* y *Postmortem*. La revisión diaria

(*Scrum Daily*) y la revisión de Sprint continua siendo una actividad sin cambios provenientes de Scrum, de esto modo se permite para detectar y remover posibles contratiempos que impidan el avance normal del proyecto. La Retrospectiva estará apoya por las fase de *Postmortem* de PSP

La fase de *Entrega*: Establece la integración del nuevo incremento con el incremento generado en el anterior *Sprint*. Se realizan las pruebas, que establezca la organización, a la liberación para verificar su funcionamiento en diferentes entornos. Un punto importante de esta fase es que la documentación del proyecto tiene que estar terminada y verificada.

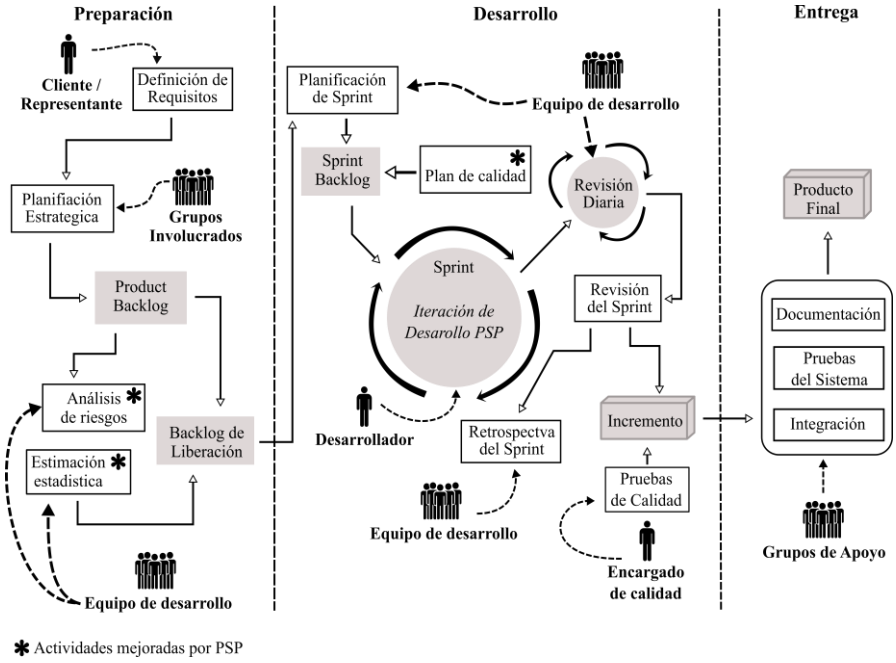

Fig. 3. Ciclo de vida de Mixing Scrum-PSP.

Iteración de Desarrollo. Es el proceso en el cual cada desarrollador utiliza las actividades clásicas para construir los programas o módulos correspondientes a sus actividades asignadas, ubicada en el *Sprint*. El marco de trabajo de PSP se centra en tres procesos: *diseño*, *código* y *pruebas*; pues son las actividades que más tiempo consumen del desarrollo del software [3]. El flujo de trabajo (Figura 4) está basado en: PSP 2.1, la iteración consiste en 8 actividades descritas a continuación:

- *Planificación:* Se produce un plan detallado para trabajar el desarrollo del programa definido por los requisitos del problema, los formatos para escribir el plan de trabajo no son difíciles pero requieren toda la atención del desarrollador. El plan consiste en la obtención y definición de los requerimientos para el programa escritos en documento claramente y sin ambigüedades, y una buena estimación de tiempo para completar el desarrollo del programa, apoyado por el método *PROBE*.

- *Diseño Detallado:* Se realiza un diseño detallado para las especificaciones del programa definido por los requerimientos, las herramientas utilizadas es responsabilidad del desarrollador.
- *Revisión de Diseño:* Validación de la consistencia del diseño, estrategia de verificación y detección de errores.
- *Código:* La transformación del diseño a sentencias de lenguajes de programación.
- *Revisión de Código:* Examinar la calidad del código mediante la detección temprana de errores.
- *Compilación:* Se traducen las sentencias del lenguaje de programación a código ejecutable. La mayoría de los defectos de sintaxis serán removidos durante esta fase. Esta *fase* es *opcional*, determinada por el entorno de desarrollo y el lenguaje de programación.
- *Pruebas Unitarias:* Cada desarrollador realiza pruebas unitarias al programa o módulo para verificar que cumpla con los requerimientos, no se establece un número límite para las pruebas o herramientas para realizarlas, sin embargo se tienen que registrar el tipo de cada prueba realizada.
- *Postmortem:* Puede considerarse una retrospectiva personal para resumir y analizar los datos generados por el proceso. Estos datos incluyen valores sobre el tiempo estimado y el tiempo real utilizado, calidad y productividad. Además, la información proporcionada por los productos personales permite dar bases a las retrospectivas del *Sprint*.

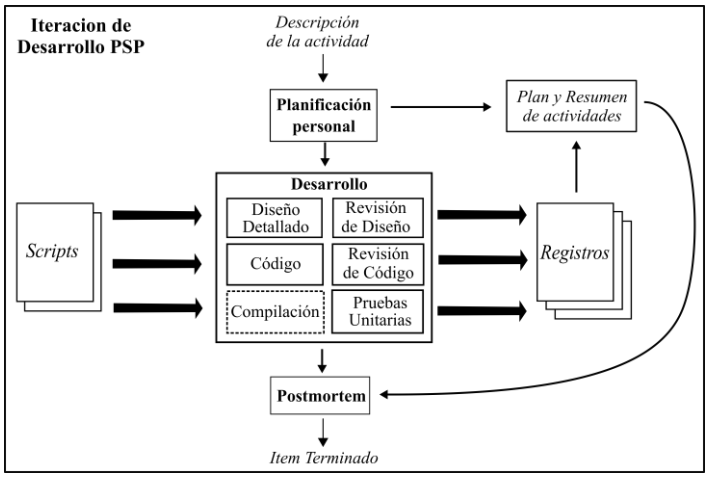

Fig. 4. Iteración de trabajo PSP.

5. Caso de Estudio

Para verificar el modelo de integración y su factibilidad se establece un caso de estudio con el propósito de implementar y documentar la experiencia de combinar dos enfoques para el desarrollo de software, Scrum y PSP, con la finalidad de mejorar la calidad del proceso de desarrollo a través de la introducción de actividades disciplinarias para mejorar las habilidades

de los desarrolladores de software. Por el cual se realiza la siguiente pregunta de reflexión: *¿En qué medida la combinación de Scrum y PSP mejora la calidad del producto final de manera indirecta mejorando la gestión del proceso de construcción?*

Se cuenta con un equipo de desarrollo conformado por tres integrantes dentro de una empresa dedicada al desarrollo de software de telemática, todos tienen conocimientos básicos sobre el método de desarrollo Scrum. La capacitación se realizará por medio de video-tutoriales alojados en una plataforma para aprendizaje en línea, tomando como referencia los materiales proporcionados por el *SEI*.

Para la recolección de datos se utilizarán las herramientas *CONFLUENCE* y *JIRA*, ambas de Altlassian, para la gestión de requerimientos y la gestión de proyectos respectivamente, estas dos herramientas soportan el desarrollo ágil de Scrum. Para la recolección de los datos de PSP se tiene el uso de métodos cuantitativos con énfasis en conocer el rendimiento personal de desarrollador proporcionados por la herramienta *PSP Dashboard*, de Tuma Solutions.

6. Conclusiones y trabajo futuro

En esta investigación se ha presentado una integración de Scrum con PSP preservando la esencia de ambos enfoques que mejora el trabajo individual de los integrantes del equipo introducción nuevas prácticas que fomentan la disciplina del proceso y la institucionalización de mejores prácticas trabajo de la empresa para el desarrollo de software. Una recomendación para la Comunidad Ágil es la extensión de métodos ágiles inspirados en las metas y expectativas que proporcionen los métodos disciplinados que prometen reforzar las debilidades de ambos enfoques.

Como trabajo futuro para esta investigación se debe realizar la comparación de los resultados de implementación con otros métodos de desarrollo de software para crear una perspectiva de los beneficios y debilidades que ofrece el modelo. Mejorar la integración estableciendo un análisis de riesgos similar a [12] para generar un modelo sólido y personalizado para cada proyecto. Enriquecer el modelo con la adicción de normas de calidad como: *ISO/IEC*, niveles de *CMMI-DEV*, prácticas de *MoProSoft* o *CompetiSoft*. Enriquecer la dinámica del trabajo en equipo con prácticas de *Team Software Process* (TSP).

Agradecimientos. Esta investigación fue apoya por El Consejo Nacional de Ciencia y Tecnología (CONACYT) bajo el Programa Nacional de Posgrados de Calidad (PNPC) y el Instituto Tecnológico de Orizaba (ITO) División de Estudios de Posgrado e Investigación (DEPI).

Referencias

1. Boehm, B., Turner, R.: Balancing agility and discipline. Addison-Wesley, Boston (2004).
2. Palacio, J., Ruata, C.: Scrum Manager Gestión de Proyectos. SafeCreative, 4th ed., Disponible en: http://www.safecreative.org/work/1012268137397, pp. 57–87 (2010)
3. Palacio, J.: Flexibilidad con Scrum. 2nd ed., Disponible en: http:// www.safecreative.org/work/0710210187520, pp. 87, 125–176 (2007)
4. Humphrey, W.: PSP: A Self-Improvment Process for Software Engineers. Upper Saddle River, NJ, Addison-Wesley (2005)

5. Sutherland, J., Schwaber, K.: The Definitive Guide to Scrum: The Rules of the Game. Scrum, Org and Scrum Inc. (2014)
6. Boehm, B.: Get ready for agile methods, with care. Computer, 35(1), pp. 64–69 (2002)
7. Paulk, M.: Agile Methodologies and Process Discipline. CrossTalk: The Journal of Defense Software Engineering, 15(10), pp. 15–18 (2002)
8. Sutherland, J.: Scrum and CMMI Level 5: The Magic Potion for Code Warriors. AGILE (2007)
9. Nawrocki, J.: Toward maturity model for extreme programming. Proceedings 27th EUROMICRO Conference, A Net Odyssey (2001)
10. Dzhurov, Y.: Personal Extreme Programming – An Agile Process for Autonomous Developers. In: Proceedings of the International Conference on Software, Services & Sematic Technologies (2009)
11. Brown, D.: PSP Implementations for agile methods: a SEMAT-based approach. Software Engineering: Methods, Modeling, and Teaching, 3, pp. 41–45 (2014)
12. Boehm, B., Turner, R.: Using risk to balance agile and plan- driven methods. Computer, 36(6), pp. 57–66 (2003)