UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

TIA
Tecnología, Investigación y Academia

Scrum versus XP: similitudes y diferencias

Scrum vs XP: Similarities and Differences

Juan Camilo Salazar[1], Álvaro Tovar[2], Juan Carlos Linares[3], Alexander Lozano[4], Lizeth Valbuena[5]

Para citar este artículo: Salazar, J. Tovar, A., Linares, J. Lozano, A., Valbuena, L. (2018). Scrum contra XP: similitudes y diferencias. *TIA*, *6*(2), pp.29-37.

Resumen

El impacto de los principios ágiles en el desarrollo de *software* ha originado el surgimiento de diferentes metodologías y corrientes. Entre las metodologías más conocidas se encuentra a Scrum y XP, las cuales han tenido una amplia aceptación y aplicación en la industria, aunque por partir de la familia de las metodologías ágiles han tendido a ser confundidas como una misma metodología; de acuerdo con ello, este artículo pretende identificar las similitudes y diferencias entre ambas metodologías, al igual que suministrar una propuesta acerca de cómo pueden ser empleadas de forma conjunta.

Palabras clave: *software*, metodologías ágiles, Scrum, *eXtreme Programming* (XP).

Abstract

The impact of agile principles in software development has led to the emergence of different methodologies and trends. Scrum and XP are among the most well-known methodologies, which have been widely accepted and applied in the industry, although have been confused as a single methodology because starting from the agile methodologies family. According to this, this paper aims to identify the similarities and differences between both methodologies, as well as provide a proposal about how they can be used jointly.

Keywords: software, agile methodologies, Scrum, eXtreme Programming (XP).

Artículo de investigación
Fecha de recepción:
31-05-2016
Fecha de aceptación:
08-06-2018

ISSN: 2344-8288
Vol. 6 No. 2
Julio - Diciembre 2018
Bogotá-Colombia

[1] Ingeniero de Sistemas, Especialista en Ingeniería de Software, Universidad Distrital Francisco José de Caldas. Ingeniero de Proyectos, Pontificia Universidad Javeriana. Correo electrónico: jcsalazarr@correo.udistrital.edu.co
[2] Universidad Distrital Francisco José de Caldas. Correo electrónico: alvaro.tovarred@gmail.com
[3] Universidad Distrital Francisco José de Caldas. Correo electrónico: juanklg25@gmail.com
[4] Universidad Distrital Francisco José de Caldas. Correo electrónico: alexander.lozanof@gmail.com
[5] Ingeniera de Sistemas, Universidad de Cundinamarca; Especialista en Ingeniería de Software, Universidad Distrital Francisco José de Caldas. Profesional Enterprise Service Bus, Banco Popular. Correo electrónico: yudylizeth@gmail.com

INTRODUCCIÓN

En la actualidad es de vital importancia crear estrategias que contribuyan al lanzamiento de productos de *software* orientados a la entrega temprana de resultados tangibles, además de la respuesta ágil y flexible a las condiciones impuestas en mercados que presentan una evolución constante; frente a ello, el modelo de construcción de *software* que se ha adoptado en estos tiempos consiste en construir un producto mientras se modifican y aparecen nuevos requisitos. Un proyecto de *software* inicia con una visión medianamente clara por parte del cliente, que en ocasiones difiere respecto al producto final dado el nivel de innovación aplicado y la velocidad a la que se mueve el sector de negocio. Al final ya no se tiene un "producto final", sino una continua evolución y mejora de una visión inicial de *software*; por ello, "La gestión de proyectos ágil no se formula sobre la necesidad de anticipación, sino sobre la de adaptación continua" [1].

Con base en lo anterior, el 17 de febrero del 2001, a petición de Kent Beck se reúnen diecisiete expertos en modelos de *software* basados en procesos (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 Cecil Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland y Dave Thomas) [2] con el fin de llegar a un acuerdo uniendo sus puntos de vista acerca de las nuevas metodologías de desarrollo para entonces denominadas ligeras. El resultado de este encuentro fue un documento que contenía los cuatro postulados denominados "Manifiesto ágil", los cuales resumen los valores sobre que se fundamentan estos métodos.

Sin disminuir la importancia en la calidad de los productos de *software* o llegar a demeritar el valor de los procesos y la documentación, este manifiesto planteaba enfocarse de manera más sustancial en los individuos e interacciones sobre procesos y herramientas, el *software* funcionando sobre documentación extensiva, la colaboración con el cliente sobre negociación contractual y la respuesta ante el cambio sobre seguir un plan; de tal manera que estos postulados contribuyeran al desarrollo rápido de aplicaciones de alta calidad.

Desde el surgimiento del Manifiesto ágil en 2001 hasta la década actual, ha habido una evolución explosiva en la aplicación de los principios ágiles en diversas industrias, desde el *software* hasta en la administración; dicha evolución ha ido de la mano con la creación de diversas metodologías que aplican de una manera particular los principios definidos, entre los que se puede encontrar al *eXtreme Programming* (XP), Scrum, desarrollo *software Lean*, desarrollo guiado por características (*Feature Driven Development*), y metodología Crystal [3].

Cada una de estas propuestas metodológicas ha traído consigo propuestas y prácticas que tienen como columna central los principios ágiles del manifiesto, y a su vez afirman ser eficaces, en especial para proyectos en problemas [3]. Entre estas propuestas se encuentra Scrum y XP como metodologías ampliamente aplicadas y extendidas tanto en el ámbito académico como en las empresas y departamentos de desarrollo; sin embargo, la diferencia entre una y otra metodología en ocasiones no es clara, por ello, a continuación, se expondrá la diferencia entre ambas metodologías, sus similitudes y cómo aplicarlas de una forma efectiva.

CARACTERÍSTICAS GENERALES DEL AGILISMO

Alrededor de los 90 empezaron a surgir movimientos y prácticas que se identificaron como metodologías livianas, con el fin de mejorar los procesos de desarrollo de *software*, estos procesos desde sus inicios han tenido que enfrentar problemas que surgen de la inclusión de nuevas tecnologías hasta cambios sociales o las necesidades cambiantes donde cada vez se

abordan proyectos más complejos y es necesario coordinar el trabajo conjunto de equipos y disciplinas diferentes [4].

Por ese entonces era muy común escuchar del modelo en cascada, un modelo basado en un ciclo convencional de ingeniería con una visión muy simple en el cual se realiza el desarrollo de *software* siguiendo una serie de etapas o fases donde se realizan tareas específicas recorriendo el ciclo de vida del *software*; el modelo propone etapas que van desde el análisis hasta el mantenimiento después de la entrega del *software* y por su gran sencillez y pasos tan intuitivos a la hora de desarrollar un *software* es implementado en muchos proyectos. Aunque parece sencillo el modelo, es importante entender la realidad de los proyectos de *software* y el contexto en el cual se aplica, los proyectos suelen tener problemas que hacen que el flujo secuencial de tareas no se realice de acuerdo al plan y en la mayoría de los casos los requerimientos de los clientes no están definidos explícitamente y generan incertidumbre en el desarrollo. También es muy común que en el desarrollo surjan errores que solo se perciben cuando el *software* ya está funcionando y, en el caso, de ser un error de diseño o de alguna de las fases tempranas del desarrollo, es necesario retroceder y puede ser desastroso para el proyecto modificando los tiempos y generando incomodidades al cliente.

Profesionales de *Standish Group* se dedicaron a obtener información de los proyectos fallidos en tecnologías de la información (IT, por sus siglas en inglés) y así poder encontrar y combatir las causas de los fracasos. En 1994 obtuvieron los siguientes resultados en 50 000 proyectos estudiados:

- Porcentaje de proyectos que son cancelados 31 %.
- Porcentaje de proyectos problemáticos 53 %.
- Porcentaje de proyectos exitosos 16 %.

Los proyectos que salieron exitosos cumplieron con un alto porcentaje de la funcionalidad requerida, pero en ningún caso fue del 100 %. Estos resultados generaron la necesidad de hacer inversión por parte de grandes empresas para perfeccionar las metodologías y tecnologías en el desarrollo de *software*, pero diez años después se hizo un nuevo estudio donde los casos de éxito subieron al 29 %, lo cual seguía siendo un porcentaje muy bajo. La mayoría de estos proyectos fracasaron por malas prácticas de gestión y otros por recursos.

Aunque se intente seguir un modelo para el desarrollo de *software* siempre van a surgir problemas que dañen el proyecto y muchas de las tareas que se hacen de estimación de tiempos y planificación suelen ser en vano, resultan siendo tareas repetitivas que alargan los proyectos; es por esto que en el 2001 se reunieron en Utah (EE.UU.) un grupo de diecisiete profesionales reconocidos del desarrollo de *software* [5], y referentes de las metodologías livianas existentes al momento con el fin de dar un pare a las prácticas tradicionales. Estos individuos con su experiencia en la implementación del modelo en cascada conocían las falencias donde primero se analiza, luego se diseña y después se implementa, pretendían ofrecer una alternativa a los procesos de desarrollo de *software* tradicionales y fue así como se creó la *Agile Alliance*, una organización sin fines lucrativos que simplemente quería promover los valores y principios de la filosofía ágil por medio de algo llamado el Manifiesto ágil [2].

El manifiesto indica los aspectos que se deben valorar en el desarrollo de *software*, indica doce principios para estos valores que abarcan aspectos dentro del desarrollo de *software*. La finalidad de los principios es reducir los problemas clásicos y dar más valor a las personas que componen el equipo de desarrollo, satisfaciendo al cliente y al desarrollador. El desarrollo de *software* de forma ágil plantea que se cambie totalmente la forma de entregar un *software*, no se hace una planeación desde el principio de todo el proyecto, sino que se hace un planeación constante de iteraciones con entregables en tiempos cortos hasta el final del proyecto, es en cada una de estas iteraciones donde aún se puede ver el modelo en cascada pero aplicado a un componente del proyecto que

en caso de necesitar un cambio no afectará todo el proyecto en general. Uno de los principales objetivos es hacer que el proyecto pueda adaptarse a los cambios, un gran factor que lleva los proyectos al fracaso. En las metodologías ágiles se proponen equipos pequeños (no mayores a diez personas) y en caso de ser un proyecto grande se recomienda crear varios equipos y dividir el proyecto, esto hace que sea más fácil para los equipos autoorganizarse y reducir la carga de trabajo del equipo [6].

Cada uno de estos principios invita a ver comportamientos y prácticas que no suelen ser fáciles de adoptar, muchas personas y empresas se quedan en el intento por falta de disciplina o de una verdadera intencionalidad de trabajar con metodologías ágiles. Para lograr esto es importante buscar ayuda de personas expertas, existen muchas empresas que ofrecen la formación y el entrenamiento para adoptar las metodologías y también hay sitios en internet donde se encuentra documentación al respecto y foros donde se cuentan experiencias.

CARACTERÍSTICAS DE SCRUM

Scrum es una metodología para el desarrollo de *software* iterativa e incremental, debe su nombre a la jugada de rugby llamada de la misma manera, se dice que es iterativa ya que se ejecuta en bloques temporales cortos y fijos (de no menos de dos semanas) que reciben el nombre de *sprints* y es incremental en tanto se obtienen funcionalidades del producto final al terminar cada iteración [7].

Dentro de las características de Scrum, se relaciona que está definido en base a roles, reuniones y artefactos, los cuales se describen a continuación.

Roles

La figura del director del proyecto no está definida en Scrum, pero sus responsabilidades están divididas en tres roles fundamentales:

- El dueño de producto o *product owner* que administra el producto y la finalidad de este es quien conoce la finalidad del proyecto, es quien interactúa con el cliente y se encarga de recibir los requerimientos.
- El Scrummaster que gestiona los procesos y garantiza que el equipo cuente con todos los insumos necesarios para realizar su trabajo.
- El equipo que se encarga del desarrollo de las diferentes funcionalidades del sistema.

Artefactos

En Scrum no existen formatos que deban ser completados al pie de la letra, sino que se sugieren diferentes artefactos que permiten llevar la trazabilidad del proyecto a través del tiempo como se menciona a continuación.

- Pila del producto o *product backlog*: en este artefacto se relacionan las diferentes funcionalidades, cambio y errores del producto final, con tres características fundamentales, deben ser simples, para que puedan ser entendidos por todo los miembros del equipo, suelen usarse las historias de usuario; también deben estimarse, indicando que tan complejo puede llegar a ser su desarrollo, sin indicar su coste en tiempo y finalmente deben priorizarse para así poder desarrollar las funcionalidades de acuerdo a su importancia.
- Pila del *sprint o sprint backlog*: se presentan las diferentes tareas que deben realizarse para desarrollar una funcionalidad propuesta en el *product backlog*, en este artefacto se presenta una estimación en horas de cuánto puede tardar la realización de cada tarea, además se indica el miembro del equipo que se ofreció para realizarla y cuánto esfuerzo (tiempo) es necesario aún para terminar cada una de las tareas propuestas.
- Gráfico de avance o *burndown chart*: en este artefacto se puede apreciar el estado de avance del proyecto, permite revisar posibles desviaciones en la estimación realizada para

el *sprint*, permitiendo tomar los correctivos necesarios para así no incumplir con los objetivos planteados, se grafica la cantidad de esfuerzo (horas) pendiente por emplear contra los días del proyecto.

REUNIONES/CEREMONIAS

Durante del desarrollo de un proyecto en SCRUM se utiliza una unidad de medida de avance y estructuración, esta es conocida como *sprint*, la cual adquiere demasiada importancia durante el desarrollo del proyecto mismo, por tanto, se requiere el desarrollo de reuniones o ceremonias antes, durante y después de la evolución de cada *sprint*.

Como punto inicial del proyecto se requiere el desarrollo de una ceremonia de visión, la cual no es oficialmente reconocida como una ceremonia de Scrum, pero que sí es recomendable realizar, ya que esta se desarrolla entre los *stakeholders*, el *product owner* y el equipo de desarrollo con el fin de construir de manera colaborativa los requerimientos del proyecto, estos se consignan en un artefacto conocido como *product backlog*, donde se realiza la priorización de los requerimientos y se consigna la estimación realizada por el equipo [8].

Posteriormente, se requiere el desarrollo de la ceremonia para la planeación del *sprint*, esta es nombrada según Scrum como *sprint planing*, esta reunión se realiza con todo el equipo Scrum (*product owner*, Scrummaster y equipo de desarrollo), y como resultado de esta ceremonia se crea el artefacto *sprint backlog*, el objetivo principal de esta ceremonia se centra en desarrollar las siguientes actividades:

- (Opcional) el *product owner* modifica el *product backlog*, de acuerdo a la retroalimentación recibida en el último *sprint review*.
- El equipo desarrollo y el *product owner* acuerdan un objetivo del *sprint*.
- El equipo Scrum selecciona un subconjunto de ítems del *product backlog*, que cumplan con el objetivo.
- Por cada ítem, el equipo identifica las tareas para llevarlo a cabo (*sprint backlog*).
-

Ahora bien, durante del desarrollo de cada *sprint*, se hace necesario la realización de ceremonias de seguimiento y avance con el fin de verificar el funcionamiento del equipo de trabajo, y el estado de las actividades que se están llevando a cabo por cada integrante del equipo, este es el principal objetivo de la ceremonia diaria de sincronización también conocida como Scrum *daily meeting*.

Otro de los objetivos de esta reunión consiste en 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; durante esta reunión se pretende que cada miembro del equipo de trabajo exponga sus avances, fortalezas, debilidades y opiniones en el desarrollo diario del *sprint*. Cada integrante se encargará de responder a las preguntas:

- ¿Qué he hecho desde el último Scrum *daily meeting*?
- ¿Qué haré a partir de la culminación de la ceremonia?
- ¿Cuáles han sido los impedimentos que se me han presentado o se me presentarán para llevar a cabo mis compromisos?

Como apoyo a la reunión, el equipo cuenta con la lista de tareas de la iteración donde se actualiza el estado y el esfuerzo pendiente para cada tarea, así como con el gráfico de horas pendientes en la iteración.

Al culminar un *sprint*, se debe presentar al *stakeholder* el avance que se tiene sobre el proyecto, para esto se desarrolla la ceremonia de revisión o *sprint review*, la cual consiste en una reunión liderada por el *product owner* con apoyo del equipo de desarrollo y el Scrummaster,

en conjunto con los *stakeholders*, su objetivo principal radica en presentar al *stakeholder* el incremento desarrollado en el *sprint*, para eso se siguen las actividades:

- El equipo Scrum expone cuál fue el objetivo del *sprint*, y los ítems del *product backlog* alcanzados.
- Se realiza una demostración del producto.
- Se recibe retroalimentación bidireccional.

Con la culminación de un *sprint* y como planeación para el inicio del siguiente se debe desarrollar una reunión de evaluación del *sprint* finalizado, esta se conoce como *sprint retrospective*; la ceremonia busca consolidar una autoevaluación del equipo Scrum con el fin de establecer las falencias detectadas en el *sprint* anterior, para esto 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, por esto en consenso buscan la respuesta de las siguientes preguntas:

- ¿Qué cosas han funcionado bien?
- ¿Cuáles hay que mejorar?
- ¿Qué cosas quiere probar hacer en la siguiente iteración?
- ¿Qué ha aprendido?
- ¿Cuáles son los problemas que podrían impedirle progresar adecuadamente?

CARACTERÍSTICAS DE XP

eXtreme Programming o XP es una metodología de desarrollo de *software* que se adapta a los postulados del Manifiesto ágil priorizando a la adaptabilidad y no el seguimiento de un plan, esta metodología fue propuesta por Kent Beck autor del primer libro sobre la materia, *Extreme Programming Explained: Embrace* Change (1999) [9].

Básicamente esta metodología se centra en la prueba y error para el desarrollo de un producto de *software* funcional, permitiendo la participación activa del cliente en todo el proceso como condición fundamental para el resultado exitoso del proceso, promoviendo el trabajo en equipo e impulsando el buen clima laboral.

El ciclo de vida establecido para este modelo consta de seis fases las cuales son exploración, planificación de la entrega (*release*), iteraciones, producción, mantenimiento y muerte del proyecto. En la fase de exploración el cliente define la especificación general del producto a requerir junto con las funcionalidades que contiene y la prioridad de cada una, la fase de planeación de entrega define las fechas en las cuales se finalizará y entregará el producto, Iteraciones.

Este modelo es aplicable a proyectos de mediano alcance para equipos de no más de veinte personas, recomienda el trabajo en parejas como alternativa de disminución de impacto en caso de eventualidades.

Valores y principios

Los valores y principios que sigue este modelo son sus principales características como se muestra a continuación.

- La comunicación: tal como se mencionó anteriormente, uno de los fundamentos principales de esta metodología es la comunicación asertiva entre los involucrados del proyecto y, por ende, el trabajo en equipo generando un buen ambiente laboral.
- La simplicidad: como es característico de las metodologías ágiles lo más importante son los deseos del cliente y sus prioridades, desarrollándolos de la manera más sencilla posible.
- La retroalimentación: la comunicación permite la construcción conjunta del proyecto y de todo el equipo en las dos direcciones de cliente hacia los desarrolladores del proyecto y viceversa.

- El coraje: todos los integrantes del equipo de desarrollo y ejecución del proyecto deben estar en la disposición y la fortaleza para enfrentar el cambio constante de los requerimientos y dar su máximo rendimiento y aprovechamiento de recursos en el desarrollo de su labor.

Prácticas

XP realiza algunas recomendaciones sobre prácticas a nivel técnico que deberían implementarse en un proyecto de desarrollo. Las prácticas son las siguientes:

- Programación de a pares: consiste en una práctica en la cual dos desarrolladores se sientan juntos a realizar una misma funcionalidad. Parte del principio que dos desarrolladores trabajando juntos generan más códigos y de mejor calidad que dos desarrolladores trabajando por aparte.
- Propiedad colectiva del código: todos los miembros del equipo son dueños del código, y no solo la persona que planteó su estructura inicial.
- Espacio de trabajo informativo: en el lugar de trabajo donde se desarrolla el proyecto siempre debe estar presente un tablero de control que muestre el estado del proyecto a diario.
- Estándares de código: manejar estándares y mejores prácticas al escribir códigos ayuda a evitar dialectos particulares de cada desarrollador, y por tanto la comprensión del código será óptima [10].
- Marcha sostenible: consiste en identificar el mejor ritmo bajo el cual el grupo de proyecto puede llegar a trabajar.
- Integración continua: permite la integración, compilación y puesta en pruebas del código desarrollado, evitando posibles problemas de calidad una vez el código sea puesto en producción.

USO CONJUNTO DE XP Y SCRUM EN PROYECTOS

De acuerdo con los planteamientos de las características de cada metodología, se logra identificar que cada una de ellas posee diferentes orientaciones y propósitos; por una parte, Scrum maneja los aspectos administrativos de un proyecto de *software*, mientras que XP va enfocado a manejar los aspectos técnicos [11]. Por tanto, es posible combinar ambas metodologías de una manera efectiva, por una parte, es posible realizar las reuniones y prácticas propuestas por Scrum (el que) aplicando las prácticas y métodos definidos por XP (el cómo).; por otra parte, es de importancia identificar situaciones en las cuales no es posible aplicar Scrum o XP, o en las cuales es aplicado en un contexto donde no es posible usarlo. Es conocido dentro de la industria del *software* de proyectos ágiles que usan Scrum y XP que han fracasado, implicando pérdidas en recursos financieros y de tiempo, llegando a dar la impresión de que las Metodologías ágiles no funcionan; sin embargo, es necesario realizar un análisis sobre las condiciones que llevan a dicho fracaso, que por ejemplo son descritas en [12].

Integración de prácticas de Scrum y XP

Para identificar cómo se relacionan las prácticas descritas por XP y Scrum, se plantea enmarcarlas en las distintas ceremonias de un proyecto. En la Tabla 1 se relacionan las ceremonias definidas en Scrum contra las prácticas que pueden aplicarse de XP.

Tabla 1. Relación de las prácticas entre Scrum y XP

Ceremonia	Práctica XP
Sprint planning	Marcha Sostenible: durante la sesión de estimación de los ítems del *product backlog*, realizar estimaciones reales y comparables contra el histórico de ejecuciones de los *sprints*. "Lo que cuenta son horas de trabajo enfocadas y motivantes" [13]
Ejecución *sprint*	Programación de a pares: fomentar entre los miembros del equipo y durante el *sprint* un 10% del tiempo para realizar *pair programming*. La idea es hacerlo de manera gradual y no forzado, ya que no es para todos. Propiedad colectiva del código: declarar como política del grupo la creación de repositorios de código fuente, además de una mayor rotación durante la programación de a pares. *Test driven development* (TDD): impulsar el uso de TDD para una parte del s*oftware* que está siendo desarrollado, por lo menos en las sesiones donde el equipo realiza *pair programming*. Integración continua: plantear un *sprint* 0 donde se pueda realizar el montaje de herramientas de integración continua, además de que permita que el equipo pueda adaptarse a su uso.
Daily Scrum	Espacio de trabajo informativo: crear en el espacio de trabajo un tablero tipo Kanban donde se muestren las tareas pendientes, en ejecución y ejecutadas, al igual que un espacio para mostrar el *burndown chart* del *sprint*.

Fuente: elaboración propia.

¿Cuándo no aplicar Scrum/XP?

Los proyectos en los cuales se describe como fracaso total el uso de metodologías ágiles tienen algunas características comunes, en su gran mayoría dichas características muestran una visión errónea acerca de los principios y prácticas introducidas por las metodologías ágiles, específicamente Scrum y XP.

De hecho, algunos proyectos aplican principios y prácticas que intentan enmascarar bajo la sombrilla de metodologías ágiles, cuando en realidad no lo son. Prácticas como las de extender el *daily* Scrum más de los quince minutos, *product owners* que no están presentes todo el tiempo o la participación casi nula de los *stakeholders* son acciones que pueden conducir al fracaso de un proyecto, sea ágil o no [12].

Por otra parte, existen algunos evangelistas que no identifican que Scrum/XP poseen algunas características que hacen que sea poco probable implementarlas en ciertos tipos de proyectos; por ejemplo, al intentar manejar varios equipos a la vez presenta la seria desventaja de que las prácticas Scrum pueden ser contraproducentes si no se adaptan al contexto que implica manejar más personas.

De acuerdo con lo anterior, en la Tabla 2 se plantean características donde cuales se aplica Scrum/XP erróneamente, y en las cuales definitivamente no se pueden aplicar.

Tabla 2. Cómo y cuándo no aplicar Scrum/XP

Clasificación	Descripción
Scrum/XP es mal aplicado cuando.	No se involucra al cliente desde el primer día, y no se hace énfasis en la participación del mismo durante los *sprint reviews*. Realizar levantamiento de requerimientos tradicional, esperando la generación de documentos y un diseño antes que desarrollo (*Big requirements up front BRUF*) [12]. Solicitar de manera explícita la generación de documentos de diseño, generando inclusive como salida de un *sprint* la creación de documentación que no genera valor al cliente.
Scrum no debería ser utilizado cuando.	Existen equipos distribuidos en varias ubicaciones y cada equipo está a cargo de distintas organizaciones/proveedores.

Fuente: elaboración propia.

CONCLUSIONES

El uso de Scrum y XP en el desarrollo de proyectos introduce un conjunto de prácticas y métodos que ayudan a mejorar el rendimiento de un equipo, además de mostrar de forma notable la evolución de un producto *software* y su calidad; sin embargo, es necesario siempre analizar el contexto bajo el cual se desenvuelve el desarrollo del proyecto con el fin de aplicar la metodología que más se ajuste a las condiciones del medio. El uso de Scrum/XP, y en general de las metodologías ágiles, puede llegar a traer grandes beneficios, pero como todo siempre dependerá del contexto en el que se apliquen y si sus principios son bien aplicados.

REFERENCIAS

[1] Scrum Manager Body of Knowledge (2014). Introducción: la agilidad. Recuperado de http://www.scrummanager.net/bok/index.php?title=Introducci%C3%B3n:_La_agilidad

[2] Beedle, M., Van Bennekum, A., Cockburn, A., Fowler, M. & Highsmith, J. (2001). Manifesto for agile *software* development. Recuperado de http://agilemanifesto.org/

[3] S. Sciencedirect, (2012). A decade of agile methodologies: towards explaining agile *software* development. *The Journal of Systems and Software*, *85*, 1213-1221.

[4] Alaimo,M y Salías, M. (2013). *Proyectos ágiles con Scrum: flexibilidad, aprendizaje, innovación y colaboración en contextos complejos*. Buenos Aires: Kleer.

[5] Cervone, H. (2011). Understanding agile project management methods using Scrum. *OCLC Systems and Services: International Digital Library Perspectives*, *27*(1), 18-22.

[6] Jurado, C. (2010). *Diseño ágil con TDD*. Recuperado de http://www.carlosble.com/downloads/disenoAgilConTdd_ebook.pdf

[7] Cohn, M. (2010). *Succeeding with Agile: Software Development Using Scrum*. Boston: Addison-Wesley.

[8] Rubin, K. (2012). Essential Scrum: a practical guide to the most popular agile process. Boston: Addison-Wesley.

[9] Beck, K. w/ Andres, C. (2005). *Extreme Programming Explained: Embrace Change*. 2nd ed. Boston: Pearson Education.

[10] Angioni, M. *et al*. (2006). Integrating XP project management in development environments. *Journal of Systems Architecture*, *52*(11), 619-626.

[11] Blom, M. (2010). Is Scrum and XP suitable for CSE development? *Procedia Computer Science*, (1)1, 1511-1517.

[12] Rajpal, M. (2016). *Lessons Learned from a Failed Attempt at Distributed Agile*. Trabajo presentado en International Conference, XP, Edimburgo, Reino Unido.

[13] Kniberg, H. (2007). *Scrum and XP from the Trenches. How We Do Scrum*. Recuperado de http://wwwis.win.tue.nl/2R690/doc/ScrumAndXpFromTheTrenchesonline07-31.pdf