Aplicación de *eXtreme Programming* en la ingeniería de software

Gabriel Reyes
Valparaíso, Chile
gabrielreyesar@gmail.com

Pablo Contreras
Valparaíso, Chile
pablo.contreras.e@mail.pucv.cl

Jean Rodríguez
Valparaíso, Chile
jean.rodriguez.h@mail.pucv.cl

***Abstract*—There are a large number of applicable methodologies in software development, categorized in traditional or agile according to their particular approach. Among the agile methodologies eXtreme Programming or XP can be found, that is featured by keeps a strong relationship with the customer and improving collaborative work with developers. This is achieved by following a series of rules that organize planning, managing, design, development and testing, based on multiple values that are intertwined to archive the proposed objectives.**

***Resumen*—Existe un gran numero de metodologías aplicables en el desarrollo de software, categorizadas en tradicionales o ágiles según su enfoque particular. Entre las metodologías ágiles se encuentra la programación extrema o XP, que se caracteriza por mantener una fuerte relación con el cliente y mejorar el trabajo colaborativo con los desarrolladores. Esto se logra siguiendo una serie de reglas que organizan la planificación, gestión, diseño, desarrollo y pruebas, basadas en múltiples valores que se entrelazan para lograr los objetivos propuestos.**

***Index Terms*—Metodología ágil, cliente, planificación, gestión, diseño, desarrollo, pruebas, valores.**

I. Introducción

En ingeniería de software, ya desde la década de los 70's se proponen las bases y utilización de modelos de desarrollo que muestra una serie de normas a seguir de manera iterativa [1], dando pie a lo hoy en día se conocen como metodologías tradicionales, con la capacidad de manejar en distintas etapas los requerimientos para entregar soluciones exitosas al mercado. Sin embargo, la dificultad de aplicar estas técnicas yace en que los proyectos cada vez poseen mayores exigencias y requieren un mayor trabajo por parte del equipo a cargo, por lo que surge la necesidad de adaptar a los desarrolladores y las tecnologías empleadas para ser mas flexibles a los cambios.

Es por esto que, años mas tarde, surgen nuevos modelos de desarrollo ágil que proponen solucionar los problemas que traen consigo el uso de las metodologías tradicionales, obteniendo conceptos de estos como el desarrollo iterativo e incremental, pero aplicados de tal forma que los requisitos y soluciones pueden variar y evolucionar con el tiempo. Las metodologías ágiles que hoy conocemos, tales como Scrum; eXtreme Programming; Kanban; Feature-Driven Development; entre otros, se enfocan en tener una comunicación abierta entre los integrantes de equipo y el usuarios, siguiendo los siguientes principios propuestos en el manifiesto ágil [2]:

- Individuos e interacciones sobre procesos y herramientas.
- Software funcionando sobre documentación extensiva.
- Colaboración con el cliente sobre negociación contractual.
- Respuesta ante el cambio sobre seguir un plan.

De estos valores, nacen los siguientes principios que satisfacen las condiciones previamente mencionadas:

- Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
- Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
- Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
- Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
- Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
- El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
- El software funcionando es la medida principal de progreso.
- Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma

indefinida.

- La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
- La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
- A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

Para que a una metodología se le considere ágil, debe cumplir con todas las condiciones mencionadas antes, aunque la importancia de cada punto mencionado sobre otros varía según la estrategia seleccionada por los equipos de trabajo y la experiencia necesite o pueda aportar al cumplimiento de la misma.

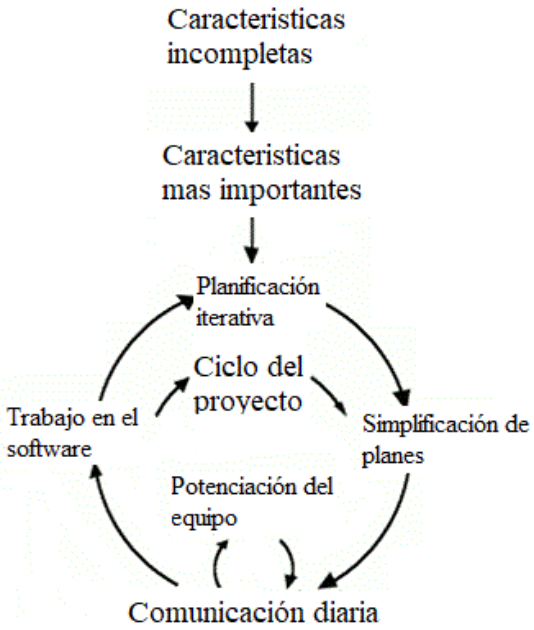

Figura 1. Diagrama de flujo ágil

Como se logra apreciar en la figura 1, para que el flujo del proceso ágil se cumpla se requiere de 4 estados evaluados por el equipo que incluyen el software y el grupo de trabajo mismo, nivelando las cargas en cada iteración. Como las personas se consideran de gran valor en el ambiente de trabajo, es necesario que sean capaces de cumplir sus roles de manera activa y eficiente, por lo que se necesita de un equipo multidisciplinario y capaz de adaptarse a los cambios que surjan de acuerdo a las necesidades del cliente.

II. Programación extrema

eXtreme Programming (o XP para abreviar) es una metodología de desarrollo ágil que aparece en los libros a finales de los 90's con la obra *Extreme Programming Explained: Embrace Change* de Kent Beck, en la cual se diferencia de los métodos tradicionales por destacar la adaptabilidad por sobre la previsibilidad. Se le considera exitosa por que enfatiza la satisfacción del cliente, y para ello se le hace entregas constantes del producto cubriendo las suficientes necesidades para que se establezca una confianza hasta el final del ciclo de vida del software, y para lograrlo se realza el trabajo colaborativo entre la administración del proyecto, el cliente y los desarrolladores [3].

En XP se implementa un entorno simple pero efectivo que permite a los equipos ser altamente productivos. El equipo se organiza de manera autónoma alrededor del problema para resolverlo de la forma más eficiente posible. Los programadores extremos se comunican constantemente con sus clientes y compañeros programadores. Mantienen su diseño simple y limpio. Obtienen comentarios al probar su software a partir del primer día. Entregan el sistema a los clientes lo antes posible e implementan los cambios sugeridos. Cada pequeño éxito profundiza su respeto por las contribuciones únicas de cada miembro del equipo.

El diagrama de flujo presentado en la figura 2 muestra cómo las reglas establecidas en *eXtreme Programming* funcionan en su conjunto. Los clientes disfrutan siendo socios en el proceso de software, los desarrolladores contribuyen de manera activa independientemente del nivel de experiencia y la gerencia se concentra en la comunicación y las relaciones, mientras se disminuyen las actividades improductivas para reducir los costos y la frustración de todos los involucrados.

III. Roles en eXtreme Programming

En este apartado se presentaran los roles que debe tener cada equipo de desarrollo de programación extrema.

Tracker: Es el encargado del seguimiento y la retroalimentación para el equipo. Además de que debe ver que tan acertadas son las decisiones tomadas y el tiempo real dedicado, también es el delegado de comunicar los resultados obtenidos para realizar futuras estimaciones.

Customer: Conocido también como cliente es quien escribe las HDU, y las pruebas para validad funcionamiento. También es quien decide que HDU se implementan en cual iteración, centrándose en agregar valor al negocio

Programmer: Es el programador, es quien escribe las pruebas unitarias y el código. Es considerado el miembro

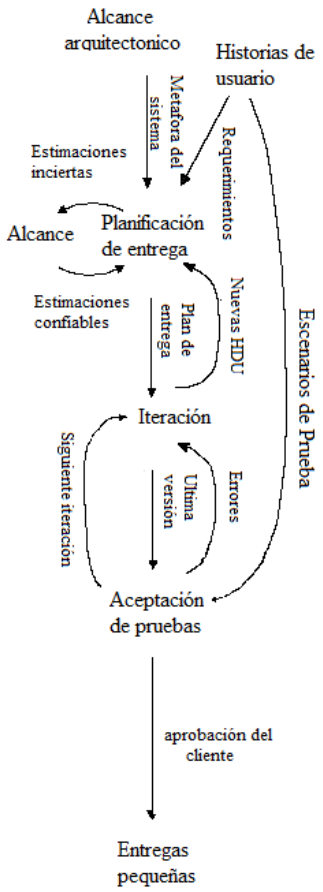

Figura 2. Diagrama de flujo XP

mas importante del equipo.

Coach: Es responsable del proceso global y se encarga de que los miembros del equipo sigan el proceso correctamente.

Tester: Es responsable del proceso global y se encarga de que los miembros del equipo sigan el proceso correctamente.

Big boss: Responsable del vinculo entre los clientes y los programadores. Su trabajo es fundamental para la coordinación.

Consultor: Es un miembro externo al equipo con un conocimiento experto en algún tema, que es necesario para el proyecto. Ayuda al equipo a resolver un problema en particular. Es posible que no siempre haya un consultor.

Manager: Encargado de agendar las reuniones necesarias, registra el resultado de las reuniones para los futuros informes del *Tracker*. Asiste a todas las reuniones.

Doomsayer: Se asegura de que todos los miembros del equipo conozcan los riesgos del proyecto.Informa a todos los riesgos que algo implica,que las malas noticias no sean ocultadas ni se ignoren y que estas no se exageren. Se debe tener cuidado al representar este rol, ya que puede bajar la productividad del equipo

Es conveniente remarcar que muchos roles pueden ser representados por mas de una persona por ejemplo alguien puede ser el *Tracker* y el *Manager*. Pero también se recomienda que otros no lo sean por ejemplo que nadie debe ser *Tracker* y *Programmer* a la vez.

IV. Reglas fundamentales de XP

La metodología XP tiene un conjunto importante de reglas y prácticas. En forma genérica, se pueden agrupar en [4]:

- Planificación.
- Gestión.
- Diseño.
- Desarrollo.
- Pruebas.

IV-A. Planificación

la planificación se plantea como un dialogo continuo entre las partes involucradas en el proyecto, incluyendo al cliente, a los programadores y a los coordinadores o gerentes. El proyecto comienza recopilando historias de usuarios (o HDU para abreviar), las que sustituyen a los tradicionales “casos de uso”. Una vez obtenidas las HDU, los programadores evalúan rápidamente el tiempo de desarrollo de cada una, y si alguna de ellas tiene “riesgos” que no permiten establecer con certeza la complejidad del desarrollo, se realizan pequeños *spikes* (programas de prueba), para reducir estos riesgos. Una vez realizadas estas estimaciones, se organiza una reunión de planificación, con los diversos actores del proyecto (cliente, desarrolladores, gerentes) con el fin de establecer un plan de entregas en los que todos estén de acuerdo. Luego de acordar este cronograma, comienza una fase de iteraciones, en dónde en cada una de ellas se desarrolla, prueba e instala unas pocas HDU.

Martín Fowler, uno de los firmantes del Manifiesto ágil, indica que los planes en XP se diferencian de las metodologías tradicionales en los siguientes aspectos [5]:

- Los planes son realizados por las mismas personas que realizarán el trabajo.
- Simplicidad del plan. No se espera que un plan requiera de expertos con complicados sistemas de administración de proyectos.

- Los planes no son predicciones del futuro, sino una estimación de cómo saldrán las cosas.

- Los planes son útiles, pero necesitan ser cambiados cuando las circunstancias lo requieren. De otra manera, se termina en situaciones en las que el plan y la realidad no coinciden, y en estos casos, el plan es totalmente inútil.

Los conceptos básicos de esta planificación son los siguientes:

Historias de usuarios: Sustituyen a los documentos de especificación funcional, y a los "casos de uso". Estas son escritas por el cliente, en su propio lenguaje, como descripciones cortas de lo que el sistema debe realizar. La diferencia más importante entre estas historias y los tradicionales documentos de especificación funcional se encuentra en el nivel de detalle requerido. Las historias de usuario deben tener el detalle mínimo como para que los programadores puedan realizar una estimación cercana y factible del tiempo que llevará su desarrollo. Cuando llegue el momento de la implementación, los desarrolladores dialogarán directamente con el cliente para obtener todos los detalles necesarios. Las HDU deben poder ser programadas en un plazo aproximado de 1 a 3 semanas. Si la estimación sobrepasa un plazo mayor entonces debe ser dividida en historias mas pequeñas, Si el tiempo estimado es menor que una semana entonces se debe combinar con otra historia.

Plan de entregas: Este plan o cronograma de entregas establece qué historias de usuario serán agrupadas para conformar una entrega, y el orden de las mismas. Esto será el resultado de una reunión entre todos los actores del proyecto, denomina acorde al tipo de empresa o cliente como *Planning game, Planning meeting* o *Planning workshop* (suelen ser sinónimos). Generalmente es cliente quien ordenará y agrupará las historias de usuario según las prioridades que presenten, entonces los desarrolladores entregarán sus plazos estimados en el plan de entregas. Luego de algunas iteraciones es recomendable realizar nuevamente una reunión con los actores del proyecto, para evaluar nuevamente el plan de entregas y ajustarlo si es necesario.

Plan de iteraciones: Las HDU seleccionadas para cada entrega son desarrolladas y probadas en un ciclo de iteración, de acuerdo al orden preestablecido. Al comienzo de cada ciclo, se realiza una reunión de planificación de la iteración. Para cada historia de usuario se establecen las pruebas de aceptación, que se realizan al final del ciclo en el que se desarrollan y al final de cada uno de los ciclos siguientes, para verificar que subsiguientes iteraciones no han afectado a las anteriores. Las pruebas de aceptación que hayan fallado en el ciclo anterior son analizadas para evaluar su corrección, así como para prever que no vuelvan a ocurrir.

Reuniones diarias de seguimiento: El objetivo de tener reuniones diarias es mantener la comunicación entre el equipo, y compartir problemas y soluciones. En la mayoría de estas reuniones, gran parte de los participantes simplemente escuchan, sin tener mucho que aportar. Para no quitar tiempo innecesario del equipo, se sugiere realizar estas reuniones en círculo y de pie.

IV-B. Gestión

En la gestión encontramos todo lo relacionado al manejo del entorno que afecta directamente a los participantes del proyecto, en donde se puede hacer mención a los siguientes puntos:

Espacios abiertos: Como la comunicación es importante en los equipos de XP, es vital mantener un área de trabajo cómodo para los integrantes del proyecto, por lo que se recomienda tomar las siguientes medidas:

- Computadores ubicados en un área central para facilitar el trabajo en pares, y mantener algunos en el perímetro para la comodidad de quienes busquen trabajar de forma individual sin perder contacto con el resto del equipo.

- Designar un área con el suficiente espacio para que se realicen reuniones diarias de sincronización del equipo sin que pierdan asistentes.

- Agregar una mesa de conferencia para tener discusiones grupales que ocurren de forma espontánea durante el día para alentar el interés de las personas a unirse.

- Tener pizarras y paredes en donde se pueda diseñar bocetos y agregar tarjetas con anotaciones para mejorar el flujo de información.

Ritmo sostenible: Esto significa mantener de forma seria el proceso de iteración que se establece para no tener que lidiar con entregas atrasadas. Para lograr esto, es necesario asignar una velocidad de progreso apropiada y que se mantenga consistente para el equipo de trabajo, lo que depende de varios factores como experiencia, complejidad del sistema, tecnologías aplicadas, entre otros.

Reuniones diarias: Los asistentes participan y no solo se quedan escuchando resultados, para que así no se pierda tiempo valioso para los desarrolladores. Cada persona discute al resto por lo menos lo siguiente:

- lo que se logro ayer.
- lo que se intentará hoy.
- los problemas que causan demoras.

Todo esto con los participantes de pie y formando un circulo, evitando largas discusiones.

Movilizar a las personas: Generar un ambiente que mantiene en movimiento a las personas logra facilitar la comunicación entre ellos y evita que los individuos encuentren un çuello de botella.ªl programar. Las empresas consideran a menudo que sus trabajadores puedan participar de capacitación cruzada para elevar la calidad del conocimiento y permitir flexibilidad al momento de repartir las cargas del trabajo. En la programación por parejas un par uno de ellos puede intercambiarse con otra pareja para moverlo de sección.

Arreglar XP: De acuerdo a la naturaleza del proyecto, puede ser necesario que algunas reglas que se mencionan deban cambiarse o necesiten una mejora. Esto se puede lograr organizando reuniones retrospectivas para discutir sobre lo que funciona y lo que no, para así mejorar XP. Este proceso debe ser una parte normal del proyecto para implementar cambios de forma eficiente y que no desbalanceen el flujo de trabajo.

IV-C. Diseño

La metodología XP hace especial énfasis en los diseños simples y claros. Los conceptos más importantes de diseño en esta metodología son los siguientes:

Simplicidad: Un diseño simple se crea más rápido que uno complejo, por lo que se propone implementar el diseño más simple posible. Con esto en mente, se sugiere nunca adelantar la implementación de funcionalidades que no correspondan a la iteración en la que se esté trabajando.

Soluciones *spike*: con la aparición de problemas técnicos o dificultades al momento de estimar el tiempo para implementar una HDU, pueden utilizarse pequeños programas de prueba para explorar diferentes soluciones. Estos programas sirven para evaluar una única solución, por lo que suelen ser desechados luego de finalizar la evaluación.

Cartas de clase, responsabilidad y colaboración: Las cartas CRC sirven para diseñar el sistema en sí como un equipo, permitiendo trabajar en un ambiente enfocado a las tecnologías orientadas a objetos por sobre la visión procedimental, a la vez que esto brinda mayores facilidades para incorporar un mayor numero de personas en la creación y modificación del diseño. Cada carta representa un objeto que posee un nombre y la responsabilidad que tienen, y luego se ordenan en conjunto de acuerdo a la comunicación que tengan los objetos entre sí. Esta técnica suele emplearse únicamente en sistemas complejos que lo requieran.

Recodificación: También llamada *refactoring*, consiste en reescribir parte del código de un programa sin cambiar su funcionalidad, para hacerlo más simple, conciso y/o entendible, ya que suele suceder que en etapas tardías de desarrollo se encuentran mejores maneras de realizar algo. En XP se sugiere recodificar cada vez que sea necesario, dando como resultado que en las siguientes iteraciones se facilite generar cambios o aumento de exigencias hacia la tarea a realizar. Como resumen de esta filosofía es mantener el código lo mas simple posible, pero que aun realice la misma función.

Metáforas: Este concepto se visualiza como una manera sencilla de explicar el propósito del proyecto, y guiar la estructura y arquitectura del mismo. Es muy importante que el cliente y los desarrolladores estén de acuerdo y compartan estas metáforas, para que puedan dialogar en un "mismo idioma". Una buena metáfora debe ser fácil de comprender para el cliente y a su vez debe tener suficiente contenido como para que sirva de guía a la arquitectura del proyecto. Sin embargo, dada la naturaleza de no estar a un mismo nivel el cliente con el desarrollador, ésta práctica resulta generalmente difícil de realizar.

IV-D. Desarrollo

Una serie de requisitos abarca el desarrollo del software, en donde podemos encontrar:

Cliente siempre presente: Esto no solamente ayuda al equipo de desarrollo, sino que pasa a ser directamente parte de él. Al comienzo del proyecto, el cliente debe proporcionar las HDU, Pero como que estas historias son cortas y de "alto nivel", no contienen los detalles necesarios para realizar el desarrollo del código. Estos detalles deben ser proporcionados por el cliente, y discutidos con los desarrolladores durante la etapa de desarrollo. No se requieren de largos documentos de especificaciones, sino que los detalles son proporcionados directamente por el cliente en el momento adecuado. Además de los beneficios que conllevan la comunicación continua, esto permite que el cliente pueda prevenir situaciones no deseables y mantener un mayor control de lo que sucede. En otras metodologías, las imperfecciones son detectadas de forma muy tardía en el ciclo de desarrollo del software, por lo que poder corregir los errores llega a ser muy complicado como costoso.

Estándares de codificación: La programación se debe realizar bajo formatos estándares, manteniendo un código consistente y fácil de comprender y manipular para quienes tengan acceso a él.

Pruebas unitarias: Aunque no parezca importante, comenzar a desarrollar pruebas unitarias antes del software requerido permite crear a futuro un programa mucho mas fácil y rápidamente, pues cuando se requieren cambios no es necesario reprogramar las pruebas, lo que evita tener que lidiar con los problemas de implementación. Los beneficios de realizar esta practica crecen a medida que

la complejidad del software aumenta, lo que no supone en ningún punto un desperdicio de tiempo o esfuerzo.Estas pruebas también tienen la propiedad de condicionar o dirigir el desarrollo del proyecto, en contraposición de las metodologías tradicionales donde los test se realizar al final del proyecto, o en el mejor de los casos al final de cada modulo de desarrollo.

Programación en pares: Esto supone un trabajo conjunto de 2 personas en un mismo computador. Aunque parezca contraproducente por impactar en el tiempo de entrega, realizar programación en pares entrega un producto de mayor calidad pues 2 personas supervisan un mismo código, facilitando el manejo de imprevistos o errores, a la vez que genera relaciones interpersonales estrechas que aumentan la rapidez de producción de soluciones.Las principales ventajas de la programación en pares son: Los programadores aprenden muchos mas sobre el sistema y desarrollo del software tratado, también aprenden los detalles de cada parte del código, por lo que da como consecuencia que los programas obtenidos sean de mayor calidad y mas cortos.

Integración continua: Los desarrolladores trabajan siempre con la versión mas reciente del software, lo que permite realizar cambios de manera rápida y es menos susceptible a retrasar el avance del proyecto. Para evitar errores en su aplicación, solamente una pareja de desarrolladores puede integrar su código a la vez. Además, las modificaciones y acceso al repositorio deben realizarse cuando sea posible, con no mas de 1 día de retraso, para siempre poder trabajar en el contexto de la ultima versión del sistema.

Propiedad colectiva: Para no generar efecto çuello de botella", todo el equipo puede contribuir con nuevas ideas que apliquen a cualquier parte del proyecto; añadiendo funcionalidades, corrigiendo errores, mejorar el diseño o la refactorización. Aunque parezca difícil en un comienzo o extraño desde una perspectiva externa a XP, Todos los participantes son responsables del trabajo y pueden aportar con sus pruebas unitarias de manera individual a la vez que trabajan en conjunto el software. En la practica, esto es mas seguro que poner a una persona a cargo que, por cualquier circunstancia y en cualquier momento, podría dejar el proyecto.En otras metodologías el concepto de "propiedad colectiva", puede parecer extraño ya que en estos se asume que la responsabilidad en el trabajo también es colectiva, por lo que generaría problemas que nadie se haga responsable.

IV-E. Pruebas

Las pruebas son una parte fundamental que aseguran la calidad del producto y facilitan el trabajo continuo sobre las posibles modificaciones que puedan recibir los requisitos planteados por el cliente. Las siguientes características mencionan como se trabajan las pruebas en el desarrollo de software:

Pruebas unitarias: Son uno de los factores principales que participan en XP, aunque trabajan de manera un tanto peculiar respecto a otras metodologías. Se comienza a trabajar bajo un marco de trabajo que maneje pruebas unitarias de forma automatizada; luego se prueban todas las clases del sistema omitiendo métodos getters y setters triviales; se mantienen en el mismo repositorio del sistema principal sin que una de estas pruebas se pierda. Estas pruebas permiten descubrir que partes del sistema funcionan correctamente y cuales se encuentran en un estado de riesgo respecto a errores que puedan suceder. Estas pueden juntarse en una única prueba universal que modularize la ejecución de pruebas pequeñas, arrojando los resultados pertinentes a cada uno de ellos. Las pruebas unitarias ofrecen una red de seguridad que ayudan a solidificar la implementación de requerimientos, mejorando el foco de atención que desarrollador debe poner en el trabajo.

Detección y corrección de errores: Cuando se encuentra un *bug*, éste debe ser corregido inmediatamente y se deben tener precauciones para que errores similares no vuelvan a ocurrir. Asimismo, se generan nuevas pruebas para verificar que el error haya sido resuelto.

Pruebas de aceptación: Trabajan de manera individual o colectiva, y se crean a partir de las HDU, en donde el cliente especifica los escenarios de prueba y verifica su correcta implementación. Estas pruebas deberían funcionar de manera autónoma y son consideradas como un sistema de caja negra, donde se espera que entreguen un resultado esperado que sea valido para la entrega del producto. Las HDU no son consideradas completas hasta que pasan por las pruebas de aceptación. Como se hablo anteriormente en el concepto de "propiedad colectiva", es recomendable hacer visibles los resultados de las pruebas de aceptación a todos los miembros del proyecto, de manera que todos estén informados.

V. Valores de XP

eXtreme Programming se basa en 5 distintos valores que son extendidas de manera natural respecto a las reglas analizadas previamente, que se promueven como una forma de trabajo en armonía en las interacciones personales y corporativas. Estos valores son [6]:

Simplicidad: Bajo la premisa de "hacer solo lo que sea necesario y solicitado", maximizamos el valor de la inversión. Dando pequeños pasos simples hacia un objetivo es mas fácil mitigar las fallas a medida que ocurran, haciéndolo mantenible a largo plazo por costos razonables.

Comunicación: Todos somos parte del equipo y nos comunicamos cara a cara todos los días, lo que permite un trabajo conjunto desde los requisitos hasta el código. Practicando una buena comunicación podemos potenciar el desempeño realizado, con lo que se evitan problemas, ya que por lo general todos los problemas que se generan en proyecto

de ingeniería de software son por falta de comunicación.

Retroalimentación: Con ello consideramos un compromiso serio hacia el trabajo, demostrando que somos capaces de poner atención a los cambios requeridos de forma temprana.Para que esto funcione el cliente también debe estar incluido dando sus comentarios sobre el software con lo que se comprenden de mejor manera sus necesidades, y hacer de mejor manera en la próxima iteración. Las pruebas unitarias también son una retroalimentación permanente que tienen los desarrolladores acerca de la calidad de su trabajo.

Respeto: Todos dan y reciben el respeto merecido como parte de un equipo valioso, en donde cada uno contribuye con conocimiento y entusiasmo. Se respeta la experiencia de los clientes y vice versa.

Coraje: Ante la expectativa y el progreso real mantendremos transparencia, dejando de lado las excusas que surjan de fallar una tarea, sin importar el tiempo que se necesita para lograr el objetivo. Nos apoyamos mutuamente para adaptarnos a los cambios.

VI. Aplicabilidad

Cada metodología tiene distintos escenarios de aplicabilidad. Es decir, las distintas metodologías de desarrollo de software no funcionan en todos los proyectos.

Como antes se mencionó, XP es una metodología recomendada para proyectos medianos, y en lo que las especificaciones no se puedan obtener hasta luego de haberlos comenzado. Además, aplica a equipos pequeños, aunque si bien no hay una regla que límite el número máximo de desarrolladores.

Otro aspecto clave para llevar acabo esta metodología, es el entorno físico en que se realizan los desarrollos. Debe ser un ambiente que permita la fácil comunicación y colaboración, como requerimiento clave en la infraestructura para poder implementar XP.

Por otro lado, la participación del cliente en el proceso es fundamental. El cliente debe conocer y aprobar la metodología, y destinar los recursos necesarios. De otra manera, el proyecto no podrá realizarse.

Finalmente, el trabajo en equipo por parte de los participantes del desarrollo es un aspecto clave dentro de esta metodología, ya que deben estar compenetrados con el proyecto y en ésta. Además, deben aceptar compartir sus códigos y ser responsables por el código que escribieron otros, jugando un factor importante la documentación.

VII. Comparación XP con scrum

Para tener un mejor entendimiento final de programación extrema, se hará un analisis comparativo con una de las metodologias ágiles mas famosas y usadas en el mercado *Scrum*, Pero antes de comenzar se dará una pequeña introducción a *Scrum*

Scrum: Es una metologia agil para gestionar el desarrollo de software, cuyo principal objetivo es maximizar el retorno de la inversión para su empresa (ROI). Se basa en construir primero la funcionalidad de mayor valor para el cliente siguiendo los principios de inspección continua, adaptación, auto-gestión e innovación [7].El proceso que *Scrum* sigue es el siguiente:

- **Product Backlog**: Conjunto de requisitos denominados historias descritos en un lenguaje no técnico y priorizados por valor de negocio, osea, por retorno de inversión considerando su beneficio y coste. Los requisitos y prioridades se revisan y ajustan durante el curso del proyecto cada cierto tiempo.

- **Sprint Planning**: Reunión durante la cual el Product Owner presenta las historias del backlog por orden de prioridad. El equipo dice la cantidad de historias que pueden ser factibles a completar en ese sprint, para en una segunda parte de la reunión, decidir y organizar cómo lo iran a conseguir.

- **Sprint**: Iteración de duración prefijada durante la cual el equipo trabaja para convertir las historias del Product Backlog a las que se ha comprometido, en una nueva versión del software totalmente funcional.

- **Sprint backlog**: Lista de las tareas necesarias para llevar a cabo las historias del sprint.

- **Daily sprint meeting**: Reunión diaria informal de máximo 15 min. en la que el equipo se organiza para trabajar de forma coordinada. Cada miembro comenta que hizo el día anterior, que hará hoy y si han ocurrido problemas.

- **Demo y retrospectiva**: Reunión que se celebra al final del sprint y en la que el equipo presenta las historias conseguidas mediante una demostración del producto. Posteriormente, en la retrospectiva, el equipo analiza qué se hizo bien, qué procesos serían mejorables y discute acerca de cómo perfeccionarlos.

Con todo esto la visto, se procese a comparar *Scrum* con *Programacion extrema*,

- XP se centra en la programación y creación de producto, y scrum en la administración del proyecto.

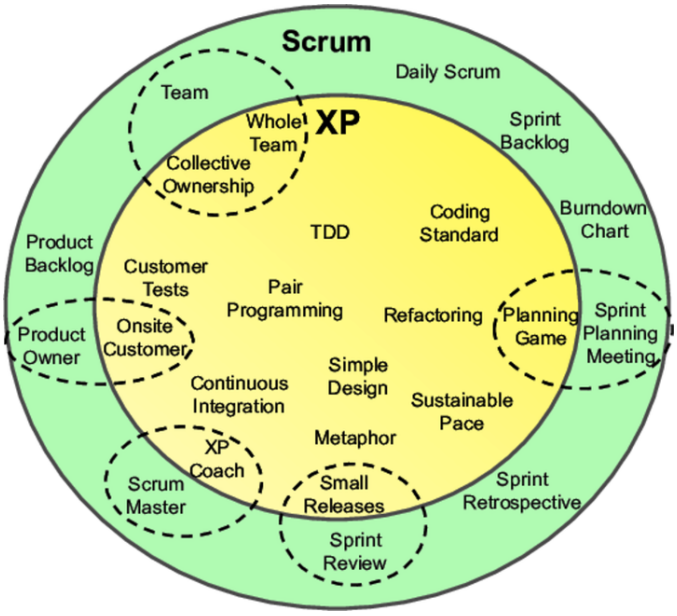

Figura 3. Origen: "https://medium.com/agile-outside-the-box/better-together-xp-and-scrum-c69bf9bffcff"

- XP sigue el orden de prioridad dado por el cliente, y scrum puede modificar el orden dado por el product owner.
- En XP se trabaja en parejas durante el proyecto, y en scrum de manera individual.
- En Xp las iteraciones son de 1 a 3 semanas, y en scrum son de 1 mes.
- En XP las funcionalidades pueden cambiar durante el proyecto, incluso si estas bien programadas y son correctas, en scrum las tareas realizadas durante el Sprint Backlog y aprobadas por el product owner no se vuelven a modificar.

VIII. Criticas a eXtreme Programming

Una de las críticas a XP es la dificultad de estimar cuánto va a costar un proyecto. Dado que el alcance del mismo no está completamente definido al comienzo, y que la metodología XP es expresamente abierta a los cambios durante todo el proceso, se torna sumamente difícil poder realizar un presupuesto previo. para empresas desarrolladoras de software, este suele ser un punto critico ya que estas, deben presupuestar (y ganar) proyectos.

Por otro lado, muchas de las prácticas sugeridas por XP son compartidas y utilizadas, de una u otra forma, en muchas otras metodología. Por ejemplo, tener un diseño simple, utilizar estándares y mantener un ritmo constante en el trabajo es claramente provechoso, sea cual sea la metodología utilizada. Pero otras practicas son discutibles, como por ejemplo, pensar únicamente en el diseño de lo que se debe entregaren el corto plazo, sin tener en cuenta lo que deberá realizarse inmediatamente después,o dentro de poco tiempo, no necesariamente es una buena decisión. Es verdad que muchas veces por intentar generalizar o intentar evitar problemas, se invierto tiempo valioso en tareas que no deberían necesitarlo. Por lo que si se hizo un buen análisis inicial se recompensa con un menor tiempo de desarrollo final.Por otro lado si se piensa que en cualquier proyecto siempre habrán problemas problemas, entonces no conviene gastar tiempo en dar generalidades o invertir grandes cantidades de tiempo evitándolos, entonces no conviene hacerlo. Estas practicas tiene muchos puntos de vistas distintos.

Otra posible crítica a XP refiere a la baja documentación, pensando en el posterior mantenimiento del sistema. Si bien durante el proyecto el equipo tiene en mente todas sus particularidades, hay que prever que pasará luego de entregado,cuando el equipo se separe, y sea necesario realizar algún cambio o mejora. XP propone la recodificación permanente, para asegurar la claridad y simplicidad del código. Sin embargo, puede pasar que aún en un código simple y claro no sea suficiente si un equipo externo al inicial necesite o quiera modificar algo o agregar una completamente nueva funcionalidad. Es posible que sea necesario mantener cierta documentación, aunque es deseable que ésta debe ser la mínima necesaria.

IX. *XP Lite*

Dada las criticas que surgen de utilizar XP en su totalidad, las empresas y equipos de trabajo deciden optar por una versión parcial de esta, la que se adapta de acuerdo a las necesidades que requieran quienes utilicen otras estrategias Para generar software. *XP Lite* es el termino asociado a esta practica, que en la practica tiende a generar un impacto similar cuando se aplica en diferentes metodologías, lo que se considera un factor de éxito hacia las compañías que buscan expandir las posibilidades de adaptación de equipos de trabajo que sufren de poca comunicación, excesivas presión, clientes hostiles, plazos cortos de tiempo, etc.

Como sugiere Ben Aveling [8], aquellos que deseen mezclar y combinar las practicas de *XP Lite* deben comenzar con:

- Jornadas de 40 horas semanales.
- Programación en pares.
- Un único sitio de trabajo.
- Integración continua.
- Diseños simples.
- Testear un programación base para el software.
- Pruebas automatizadas.
- Simulación de clientes.
- Refactorización.
- Estándares de codificación.

- Entrenamiento cruzado.
- Reuniones diarias y retrospectivas.

Principalmente los dos últimos puntos mencionados suelen ser necesarios para que los desarrolladores adquieran un entendimiento de que es y como funciona XP.

IX-A. Practicas adoptadas

Un grupo de 4 compañías de distintos ámbitos se utilizaron como casos de estudio para evaluar la aplicación de *XP Lite*, en donde las dos primeras, siendo categorizadas de menor rango, y las siguientes poseen un mayor alcance y recursos (ver tabla 4). Se logra apreciar como ninguna implementa metáforas (considerada por ellos como simple curiosidad) y solo las 2 compañías de largo alcance hacen uso parcial de entregas pequeñas, esto exclusivamente para realizar a modo de prueba [1].

	Company 1	Compan y 2	Company 3	Company 4
Company	Small software house	Small software house	Large corporate	Consultancy
Informant	Developer	Manager	Manager	Developer
On-site Customer	no	no	modified	modified
Planning Game	no	no	modified	modified
Small Releases	no	no	partial	partial
Metaphor	no	no	no	no
Simple Design	no	yes	yes	yes
Testing	no	yes	partial	partial
Refactoring	partial	yes	yes	yes
Pair Programming	no	partial	yes	yes
Collective Ownership	no	partial	yes	yes
Continuous Integration	no	yes	yes	yes
40-Hour Week	no	yes	yes	yes
Coding Standards	partial	yes	yes	yes
Whole Team	yes	yes	yes	yes
Coach	no	yes	yes	yes
Stand Up Meetings	partial	no	yes	yes
Unqualified Success	no	yes	yes	yes

Figura 4. Uso de *XP Lite* [9]

Las compañías explicaron algunos de los puntos propuestos evadidos de la siguiente manera, partiendo por la programación en pares:

- La **Compañía 1** no utilizaron programación en pares por que la gerencia no la percibio como rentable. En sus palabras: "si solo voy a obtener 4 horas-hombre de trabajo de lo que era originalmente 8 horas-hombre, no puedo ver el valor en eso".
- La **Compañía 2** considero que la programación en pares valía la pena solo para educar de manera inicial al nuevo personal, en parte por que dificultaba la revisión de desempeño individual de los desarrolladores.
- Las **Compañías 3 y 4** tenían una visión más amplia de los beneficios del emparejamiento, creyendo que aumenta tanto la productividad como la "supervivencia", algo que afecto de manera negativa a la **compañía 1**, en donde la perdida de un solo individuo desarrollador impacta significativamente en el desarrollo del proyecto, diciendo que ”el tipo que se va mañana, es el único que tiene capacitación en el middle-ware”.

Además la adopción de la jornada de 40 horas semanales, al igual que con la programación de pares, a veces fue bloqueada por las percepciones de la administración de la rentabilidad. Sin embargo, la evidencia de la **Compañía 1** sugiere que la presión excesiva en el horario puede ser una economía falsa: .El código es bizantino[2] y frágil. Y esto nos está matando. Esto nos está matando absolutamente". Un impulsor para la adopción de algo que se acerca a la Semana de 40 horas en la Compañía 2 fue la rotación de personal: "Nunca hemos forzado a los desarrolladores a trabajar horas ridículas porque se van".

IX-B. Recepción

Los ingenieros representantes del caso de estudio previo tenían una visión extremadamente positiva de XP, no eran menos positivos sobre las prácticas que ellos mismos no habían intentado. Los gerentes entrevistados tenían actitudes básicamente positivas hacia las prácticas que habían experimentado pero eran menos positivas, incluso desdeñosas, de las prácticas que no habían experimentado personalmente.

La adopción de *XP Lite* generalmente estuvo motivada no tanto por los beneficios percibidos de XP en general, sino mas bien por como aplacaba las deficiencias percibidas en las metodologías actuales. La **Compañía 1** dijo que ”lo que hemos estado haciendo ... no funcionó la última vez y no está funcionando ahora. XP no podría ser peor”, mientras que la Compañía 3 dijo que ”al escuchar sobre la solución comenzamos a entender que puede resolver los problemas que teníamos. No estábamos buscando activamente un cambio en la metodología”.

X. Conclusión

eXtreme Programming es una de las mas conocidas entre las metodologías modernas de desarrollo de software. Siendo de reciente divulgación en conjunto con las metodologías ágiles, está comenzando a ser utilizada por algunos, y criticada por otros. Es claro que no existe una metodología única para garantizar el éxito de cualquier proyecto de desarrollo

[1]Considerar el año de publicación del modelo propuesto, hoy en día es esencial incluso para proyectos *open source*

[2]Parte oriental del imperio romano (395-1453), en contexto puede hacer alusión al rechazo de ser reconocido como algo legitimo y estable

de software, y esto aplica también a XP. Toda metodología requiere de cierta adaptación al proyecto, al cliente y a la idiosincrasia de la empresa desarrolladora.

Las metodologías tradicionales han intentado abarcar la mayor cantidad de situaciones, pero exigen un esfuerzo considerable en documentación y gerenciamiento de proyecto, que muchas veces genera una sobrecarga inaceptable en proyectos pequeños y medianos. XP propone una metodología ágil y concreta, aunque requiere de una nueva menara de pensar, ver y hacer las cosas, tanto por parte de los gerentes (responsables de la rentabilidad general del proyecto), como de los desarrolladores y también del cliente. La aplicabilidad de ésta metodología a cada proyecto debería ser analizada en cada caso, teniendo en cuenta el tamaño y tipo de proyecto, así como sus ventajas y desventajas.

En el mundo laboral, las empresas permiten o impiden un uso completo de XP de acuerdo a la cultura organizacional, por lo que no es una metodología universalmente aplicable. La experiencia demuestra que XP se utiliza mayormente como un conjunto de herramientas, comúnmente denominado *XP Lite* [8] que se acoplan con técnicas propias adoptadas por los equipos, lo que suele ser un punto de partida razonable para muchas organizaciones.

REFERENCIAS

[1] W. Royce, "Managing the development of large software systems," *IEEE Wescon*, pp. 1–9, 1970.

[2] A. v. B. A. C. W. C. M. F. Kent Beck, Mike Beedle, "Manifiesto por el desarrollo Ágil de software." http://agilemanifesto.org/iso/es/manifesto.html, Febrero 2001.

[3] D. Wells, "Extreme programming: A gentle introduction." http://www.extremeprogramming.org, 1999.

[4] D. Wells, "The rules of extreme programming." http://www.extremeprogramming.org/rules.html, 1999.

[5] Addison-Wesley, "Interview with kent beck and martin fowler." http://www.informit.com/articles/article.aspx?p=20972, Marzo 2001.

[6] D. Wells, "The values of extreme programming." http://www.extremeprogramming.org/values.html, 2009.

[7] K. S. y Jeff Sutherland, "La guía definitiva de scrum: Las reglas del juego," 2017.

[8] B. Aveling, "Xp lite considered harmful?," pp. 94–103, 2004.

[9] R. Moore, "Evolving to a lighter software process: a case study." https://www.researchgate.net/publication/3942166_Evolving_to_a_lighter_software_process_a_case_study, Febrero 2001.