UNIVERSIDAD AUTÓNOMA GABRIEL RENÉ MORENO

FACULTAD DE INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN Y TELECOMUNICACIONES

INVESTIGACION U2: DESARROLLO ÁGIL

MATERIA Tecnología Web

NOMBRE Pablo Michael Tardio Ventura

CODIGO 217064957 **GRUPO** SC

FECHA 09/12/2020

DOCENTE Ing. Evans Balcázar Veizaga

EXTREME PROGRAMMING

El Extreme (o XP) Programming es una metodología de desarrollo que pertenece a las conocidas como metodologías ágiles (otras son Scrum, Kanban…), cuyo objetivo es el desarrollo y gestión de proyectos con eficacia, flexibilidad y control.

Ambos conceptos, relacionados estrechamente, son distintos. Agile es el marco de trabajo para el desarrollo del software, se hace mediante un proceso iterativo y define las prácticas y roles del equipo. Por su lado, el XP programming es una metodología basada en la comunicación, la reutilización del código desarrollado y la realimentación.

HISTORIA

La programación extrema o eXtreme Programming (XP) es un enfoque de la ingeniería de software formulado por Kent Beck, autor del primer libro sobre la materia, Extreme Programming Explained: Embrace Change (1999). Es el más destacado de los procesos ágiles de desarrollo de software. Al igual que éstos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

CARACTERISTICAS PRINCIPALES

VALORES

Los valores originales de la programación extrema son: simplicidad, comunicación, retroalimentación (feedback) y coraje. Un quinto valor, respeto, fue añadido en la segunda edición de Extreme Programming Explained. Los cinco valores se detallan a continuación:

SIMPLICIDAD

La simplicidad es la base de la programación extrema. Se simplifica el diseño para agilizar el desarrollo y facilitar el mantenimiento. Un diseño complejo del código junto a sucesivas modificaciones por parte de diferentes desarrolladores hace que la complejidad aumente exponencialmente.

Para mantener la simplicidad es necesaria la refactorización del código, ésta es la manera de mantener el código simple a medida que crece.

También se aplica la simplicidad en la documentación, de esta manera el código debe comentarse en su justa medida, intentando eso sí que el código esté autodocumentado. Para ello se deben elegir adecuadamente los nombres de las variables, métodos y clases. Los nombres largos no decrementan la eficiencia del código ni el tiempo de desarrollo gracias a las herramientas de autocompletado y refactorización que existen actualmente.

Aplicando la simplicidad junto con la autoría colectiva del código y la programación por parejas se asegura que cuanto más grande se haga el proyecto, todo el equipo conocerá más y mejor el sistema completo.

COMUNICACIÓN

La comunicación se realiza de diferentes formas. Para los programadores el código comunica mejor cuanto más simple sea. Si el código es complejo hay que esforzarse para hacerlo inteligible. El código autodocumentado es más fiable que los comentarios ya que estos últimos pronto quedan desfasados con el código a medida que es modificado. Debe comentarse sólo aquello que no va a variar, por ejemplo el objetivo de una clase o la funcionalidad de un método.

Las pruebas unitarias son otra forma de comunicación ya que describen el diseño de las clases y los métodos al mostrar ejemplos concretos de como utilizar su funcionalidad. Los programadores se comunican constantemente gracias a la programación por parejas. La comunicación con el cliente es fluida ya que el cliente forma parte del equipo de desarrollo. El cliente decide qué características tienen prioridad y siempre debe estar disponible para solucionar dudas.

RETROALIMENTACIÓN (FEEDBACK)

Al estar el cliente integrado en el proyecto, su opinión sobre el estado del proyecto se conoce en tiempo real.

Al realizarse ciclos muy cortos tras los cuales se muestran resultados, se minimiza el tener que rehacer partes que no cumplen con los requisitos y ayuda a los programadores a centrarse en lo que es más importante.

Considérense los problemas que derivan de tener ciclos muy largos. Meses de trabajo pueden tirarse por la borda debido a cambios en los criterios del cliente o malentendidos por parte del equipo de desarrollo. El código también es una fuente de retroalimentación gracias a las herramientas de desarrollo. Por ejemplo, las pruebas unitarias informan sobre el estado de salud del código. Ejecutar las pruebas unitarias frecuentemente permite descubrir fallos debidos a cambios recientes en el código.

CORAJE O VALENTÍA

Muchas de las prácticas implican valentía. Una de ellas es siempre diseñar y programar para hoy y no para mañana. Esto es un esfuerzo para evitar empantanarse en el diseño y requerir demasiado tiempo y trabajo para implementar el resto del proyecto. La valentía le permite a los desarrolladores que se sientan cómodos con reconstruir su código cuando sea necesario. Esto significa revisar el sistema existente y modificarlo si con ello los cambios futuros se implementarán más fácilmente. Otro ejemplo de valentía es saber cuando desechar un código: valentía para quitar código fuente obsoleto, sin importar cuanto esfuerzo y tiempo se invirtió en crear ese código. Además, valentía significa persistencia: un programador puede permanecer sin avanzar en un problema complejo por un día entero, y luego lo resolverá rápidamente al día siguiente, sólo si es persistente.

RESPETO

El respeto se manifiesta de varias formas. Los miembros del equipo se respetan los unos a otros, porque los programadores no pueden realizar cambios que hacen que las pruebas existentes fallen o que demore el trabajo de sus compañeros. Los miembros respetan su trabajo porque siempre están luchando por la alta calidad en el producto y buscando el diseño óptimo o más eficiente para la solución a través de la

refactorización del código. Los miembros del equipo respetan el trabajo del resto no haciendo menos a otros, una mejor autoestima en el equipo eleva su ritmo de producción.

CARACTERÍSTICAS FUNDAMENTALES

Las características fundamentales del método son:

Desarrollo iterativo e incremental: pequeñas mejoras, unas tras otras.

- Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación. Véase, por ejemplo, las herramientas de prueba JUnit orientada a Java, DUnit orientada a Delphi, NUnit para la plataforma.NET o PHPUnit para PHP. Estas tres últimas inspiradas en JUnit, la cual, a su vez, se insipiró en SUnit, el primer framework orientado a realizar tests, realizado para el lenguaje de programación Smalltalk.
- Programación en parejas: se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. La mayor calidad del código escrito de esta manera -el código es revisado y discutido mientras se escribe- es más importante que la posible pérdida de productividad inmediata.
- Frecuente integración del equipo de programación con el cliente o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
- Corrección de todos los errores antes de añadir nueva funcionalidad. Hacer entregas frecuentes.
- Refactorización del código, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
- Propiedad del código compartida: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados.
- Simplicidad en el código: es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario. La programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

La simplicidad y la comunicación son extraordinariamente complementarias. Con más comunicación resulta más fácil identificar qué se debe y qué no se debe hacer. Cuanto más simple es el sistema, menos tendrá que comunicar sobre éste, lo que lleva a una comunicación más completa, especialmente si se puede reducir el equipo de programadores.

EL EQUIPO DE UN PROYECTO XP

Los equipos de un proyecto de esta tipología y magnitud tienen normalmente las siguientes figuras y roles:

- **Clientes**: Establecen las prioridades y marca el proyecto. Suelen ser los usuarios finales del producto y quiénes marcan las necesidades.
- **Programadores**: Serán los que se encargarán de desarrollar el Extreme Programming.

- **Testers**: se encargan de ayudar al cliente sobre los requisitos del producto.
- **Coach**: Asesoran al resto de componentes del equipo y marcan el rumbo del proyecto.
- **Manager**: Ofrece recursos, es el responsable de la comunicación externa y quien coordina las actividades.

En general, no obstante, los participantes en este tipo de equipos no siempre toman un rol fijo y contribuyen con los conocimientos de cada uno en aras del beneficio colectivo.

LAS PLANIFICACIONES

Por una parte se deben planificar los plazos temporales del proyecto basándose en las exigencias del cliente. En base a las estimaciones de coste y la dificultad del proyecto se marcan las prioridades y las fechas, no siempre de forma precisa, pero sí orientativa.

Con la entrega de la planificación efectuada, se desarrolla la de la iteración en el que cada dos semanas se marca el rumbo y se entrega el software útil después de cada uno de estos periodos bisemanales. Con esto se consigue que el nivel de precisión sea mucho mayor, las estimaciones sobre los costes sean más exactas y la información mucho más transparente.

PRUEBAS

Continuamente se han de efectuar una serie de pruebas automatizadas en base a los requisitos del cliente para comprobar que todo funcione correctamente. Éstas han de hacerse de forma periódica y automática.

Con las planificaciones comentadas anteriormente se incluyen las entregas al final de cada iteración, éstas serán siempre con el software probado y funcionando correctamente y será facilitado al cliente, que puede utilizarlo para cualquier propósito, incluso para el usuario final. Los equipos XP también pueden hacer entregas a otros usuarios finales.

DISEÑO Y PROGRAMACIÓN

El diseño del programa suele ser simple y basado en la funcionalidad del sistema y se lleva a cabo durante todo el proyecto, tanto durante la planificación de la entrega como en el de la iteración.

La programación del software se hace siempre en pareja, lo que se llama programar a dos manos. Se asegura con este método que al menos un programador conoce y controla la labor de otro y queda revisado. La ventaja es que se produce mejor código que en base a un programador aunque la dificultad de la misma sea mayor.

El código es de todos, con el desarrollo de las pruebas automáticas y la programación a dos manos se incluye también la posibilidad de que cualquiera pueda añadir y retocar parte del código, aunque eso sí, deba ser un estilo común y cuyo resultado sea como si sólo lo hubiera hecho una persona.

Uso de metáforas y otras ventajas

Se buscan frases o nombres que definan parte del programa para qué todos sepan a qué se refieren y cuál es su funcionalidad. Por ejemplo está el “recolector de basura” de Java.

El Extreme Programming tiene como gran ventaja el de la programación organizada y planificada para que no haya errores durante todo el proceso. Los programadores suelen estar satisfechos con esta metodología. Es muy recomendable efectuarlo en proyectos a corto plazo.

PROYECTOS DE APLICACION

MICROSOFT

El gigante de desarrollo de software Microsoft ha adoptado prácticas de desarrollo ágil desde los tiempos del lanzamiento del SQL Server 2005, según uno de sus Vicepresidentes, la adopción de ágil no fue una imposición (desde los altos niveles), sino que se realizó por la vía de incentivar y alentar. Se comenzó con prácticas de Scrum y de Extreme Programming (XP).

En lugar de adoptar todas las prácticas del principio, se comenzó por tomar algunas ideas, por ejemplo de Scrum se tomo la de reunir a todo el equipo todos los días por media hora, decidir que van a hacer y luego hacer el trabajo de forma rápida. De Extreme Programming (XP) se adoptó la de programación en pares, aplicando el concepto que dos mentes trabajan mejor encontrando los problemas más rápido.

Microsoft aprendió que en vez de establecer procesos de desarrollo de software que deben ser seguidos al pie de la letra, es mejor establecer niveles de calidad, y otorgarle a cada equipo la flexibilidad de alcanzar esos resultados de la forma que sea más efectiva para cada uno.

Como se ha dicho, la adopción no fue obligatoria sino voluntaria, y los resultados fueron variados, algunos exitosos y otros no. Anexo se presenta un link a un paper sobre un estudio exploratorio de uso y percepción del desarrollo de software.

SCRUM

FUNDAMENTOS

Scum se basa en:

• El desarrollo incremental de los requisitos del proyecto en bloques temporales cortos y fijos (iteraciones de un mes natural y hasta de dos semanas, si así se necesita).

• La priorización de los requisitos por valor para el cliente y coste de desarrollo en cada iteración.

• El control empírico del proyecto. Por un lado, al final de cada iteración se demuestra al cliente el resultado real obtenido, de manera que pueda tomar las decisiones necesarias en función de lo que observa y del contexto del proyecto en ese momento. Por otro lado, el equipo se sincroniza

diariamente y realiza las adaptaciones necesarias.

• La potenciación del equipo, que se compromete a entregar unos requisitos y para ello se le otorga la autoridad necesaria para organizar su trabajo.

• La sistematización de la colaboración y la comunicación tanto entre el equipo y como con el cliente.

• El timeboxing de las actividades del proyecto, para ayudar a la toma de decisiones y conseguir resultados.

CARACTERISTICAS

- Scrum es un marco de trabajo de procesos que ha sido usado para gestionar el desarrollo de productos complejos desde principios de los años 90. Scrum no es un proceso o una técnica para construir productos; en lugar de eso, es un marco de trabajo dentro del cual se pueden emplearnvarios procesos y técnicas. Scrum muestra la eficacia relativa de las prácticas de gestión de producto y las prácticas de desarrollo de modo que podamos mejorar.
- Scrum es un modelo de referencia que define un conjunto de prácticas y roles, y que puede tomarse como punto de partida para definir el proceso de desarrollo que se ejecutará durante un proyecto. Los roles principales en Scrum son el ScrumMaster, que mantiene los procesos y trabaja de forma similar al director de proyecto, el ProductOwner, que representa a los stakeholders (clientes externos o internos), y el Teamque incluye a los desarrolladores.
- La colaboración, ya que es una de las mejores cosas que tiene Scrum es que se fomenta mucho la colaboración entre los miembros del equipo de desarrollo, entre el cliente y el equipo.
- La auto organización, porque Scrum está muy enfocado a que los equipos sean capaces de organizarse, de auto gestionarse, de saber llevar la carga de trabajo en todo momento, de que se tenga el control del tiempo, etc. Por todo eso, los equipos son los que se marcan sus ritmos y el desarrollo es progresivo, ya que se va avanzando a medida que va pasando el tiempo y el equipo va adquiriendo madurez y se procede a un valor incremental.
- La priorización, debido a que existen prioridades y se establecen criterios para saber qué trabajos son los que lo más importante y son los primeros que hay que desarrollar. Scrum es una metodología muy abierta, flexible y que se adapta a las necesidades de los clientes en

cada momento. Si aparece algo urgente, se le puede asignar una prioridad y el equipo puede ponerse a trabajar de forma inmediata.

ROLES

por muchas razones diferentes Una fuente probable de la oposición a la adopción de Scrum es que la confusión sobre los nuevos roles que existen en un proyecto Scrum, y el papel que estos desempeñan, los roles usuales son los siguientes:

Roles Principales:

Estos roles son los que tienen más riesgo ya que son los que están involucrado de manera más profunda con el proyecto y su responsabilidad es más grande, por lo tanto cualquier falla recae principalmente sobre ellos

1.EL SCRUMMASTER

El Scrum es facilitado por un ScrumMaster, cuyo trabajo primario es eliminar los obstáculos que impiden que el equipo alcance el objetivo del sprint. El ScrumMaster no es el líder del equipo (porque ellos se auto-organizan), sino que actúa como una protección entre el equipo y cualquier influencia que le distraiga. El ScrumMaster se asegura de que el proceso Scrum se utiliza como es debido. El ScrumMaster es el que hace que las reglas se cumpla.Por ejemplo, un ScrumMaster no tiene el poder de despedir a un miembro del equipo, en cambio puede tomar decisiones como ampliar un sprint a dos Semanas en el siguiente Mes.

PROYECTO DE APLICACION

3.CASO SPOTIFY

El primer punto es que en Spotify, como cultura, se valora Agilidad más que Scrum, y a los principios ágiles más que a cualquier práctica. Eso les permite abrir la mente y no actuar con “pasión exagerada” a la hora de tomar decisiones.

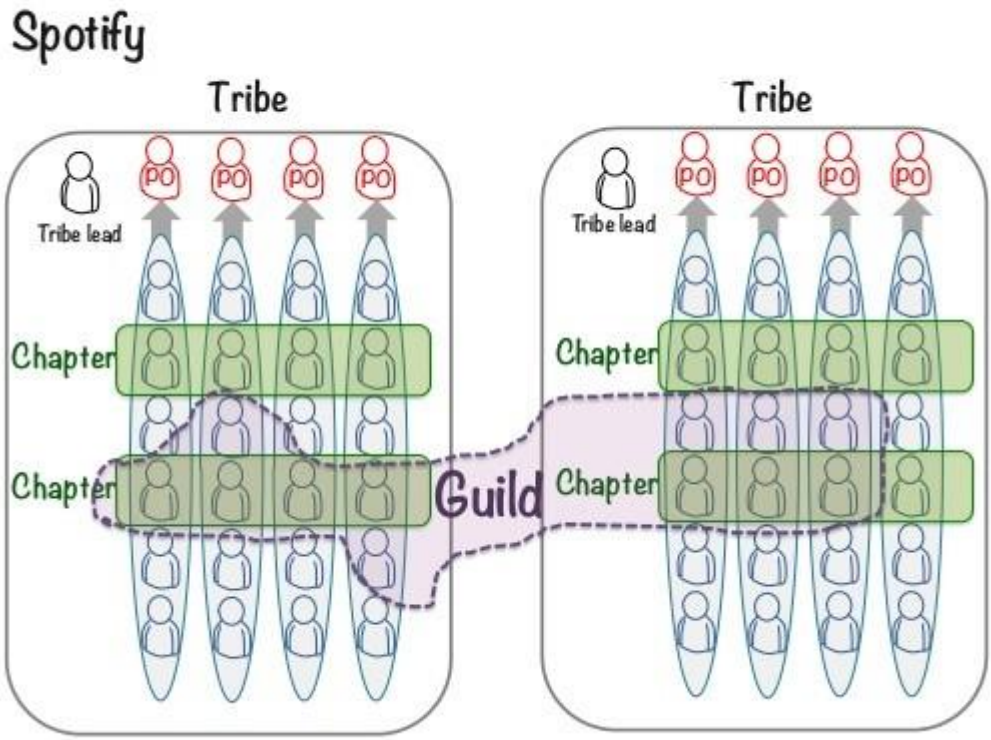
1. SCRUM MASTER COMO AGILE COACH

Spotify entiende que la adopción de un paradigma tan disruptivo como lo es la Agilidad en general y Scrum en particular, conlleva cambios culturales importantes, así como de mentalidad y de la forma de trabajar.

Es por esto que hace especial hincapié en la figura del Scrum Master como un Agile Coach, y lo escala a nivel organizacional incluyendo a las áreas que no están directamente relacionadas con el desarrollo del producto.

2. EQUIPOS PEQUEÑOS "TOTALMENTE" INDEPENDIENTES

Spotify trabaja con equipos que en ocasiones están distribuidos internacionalmente. Sin embargo ha logrado organizarlos en equipos pequeños llamados Squads (Escuadrones) a los cuales trata como pequeñas compañías que son responsables de una parte específica del producto que se desarrolla.



Para salvar las dependencias que inevitablemente existen entre esos equipos, ha creado otros tipos de estructuras a las que llama: Tribus, Consejos y Gremios, respectivamente.

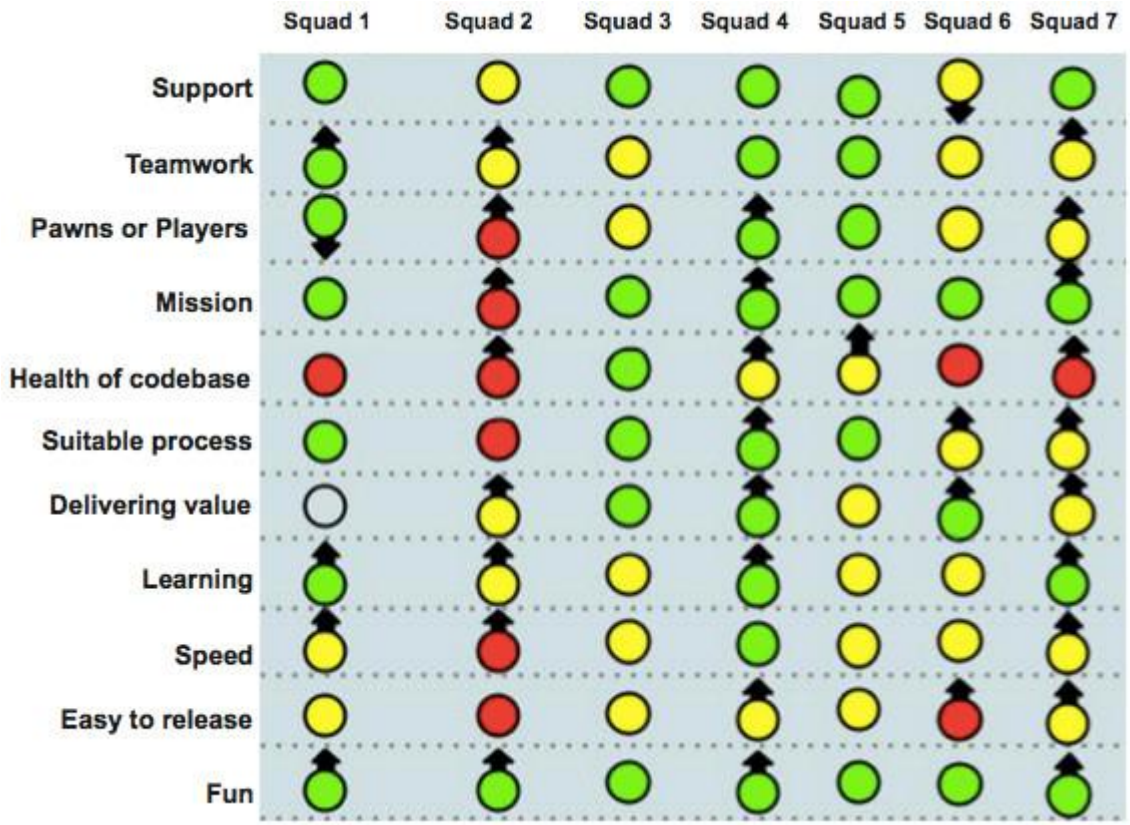
Las Tribus son agrupaciones de Escuadrones con objetivos relacionados, mientras que los Consejos y los Gremios son organizaciones transversales a nivel de Escuadrones para el caso de los Consejos, y de Tribus para el caso de los Gremios.

Los Consejos y Gremios agrupan a integrantes de todos los equipos teniendo en cuenta temáticas afines. Eso les permite mantenerse al tanto de las decisiones y del estado de los desarrollos de toda la organización. De esa manera permanecen alineados pero autónomos.

3. CREATIVIDAD EN LAS RETROSPECTIVAS

En un negocio tan creciente como Spotify, los impedimentos surgen como las cabezas de la Hidra de Lerna.

Spotify lidia con esos impedimentos monitoreando constantemente los obstáculos potenciales y eliminándolos antes de que estos se conviertan en problemas mayores.



Uno de los casos más documentados sobre las soluciones a sus problemas es el siguiente:

En una ocasión notaron que su equipo de operaciones estaba actuando muy lento para desplegar sus soluciones. Entonces decidieron eliminar ese impedimento transfiriendo la responsabilidad del despliegue de las soluciones al propio equipo de desarrollo. Los encargados de operaciones solo apoyan con herramientas y procedimientos.

Esa decisión no suele tener sentido en otras organizaciones, sin embargo, ellos determinaron que debían ser pragmáticos, abrir la mente y ser disruptivos, así han logrado el éxito hasta hoy.