**Introducción.**

La mayoría de los programadores tenemos cierta tendencia en embebernos en cuestiones técnicas, hablar de lenguajes de programación, de técnicas de programación, de entornos de desarrollo o de editores de recursos. Pero se nos pasan por alto temas muy importantes que nos afectan tanto o más que las cuestiones mencionadas, como es la ingeniería de software, la manera en que debemos de hacer nuestro software. Alrededor de cómo hacer software hay una gran numero de autores teorías, propuestas, en esta oportunidad les presentare una nueva disciplina de desarrollo de software, la programación extrema.

XP (eXtreme Programing) nace como nueva disciplina de desarrollo de software aproximadamente en 1997 y ha causado un gran revuelo entre el colectivo de programadores del mundo. Kent Beck, su autor, es un programador que ha trabajado en múltiples empresas y que actualmente lo hace como programador en la conocida empresa automovilística DaimlerChrysler. Con sus teorías ha conseguido el respaldo de gran parte de la industria del software y el rechazo de otra parte.

La programación extrema se basa en la simplicidad, la comunicación y el reciclado continuo de código.

**¿En qué consiste XP?**

Los objetivos de XP son muy simples: la satisfacción del cliente y el segundo objetivo es potenciar al máximo el trabajo en grupo.

**Desarrollo.**

Es la metodología de desarrollo de la ingeniería de software más destacada de los procesos ágiles de desarrollo de software. Al igual que éstos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que hace más énfasis en la adaptabilidad que en la previsibilidad.

**CARACTERÍSTICAS DE LA METODOLOGÍA XP**

1. Se diferencia de las metodologías tradicionales principalmente en que hace más énfasis en la adaptabilidad que en la previsibilidad.
2. Se aplica de manera dinámica durante el ciclo de vida del software.
3. Es capaz de adaptarse a los cambios de requisitos.
4. Los individuos e interacciones son más importantes que los procesos y herramientas.

La gente es el principal factor de éxito de un proyecto software. Es más importante construir un buen equipo que construir el entorno. Muchas veces se comete el error de construir primero el entorno y esperar que el equipo se adapte automáticamente. Es mejor crear el equipo y que éste configure su propio entorno de desarrollo en base a sus necesidades.

1. Software que funcione es más importante que documentación exhaustiva.
2. Desarrollar software que funciona más que conseguir una buena documentación.

La regla a seguir es no producir documentos a menos que sean necesarios de forma inmediata para tomar una decisión importante. Estos documentos deben ser cortos y centrarse en lo fundamental.

1. La colaboración con el cliente es más importante que la negociación de contratos.
2. La colaboración con el cliente más que la negociación de un contrato.

Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito.

1. La respuesta ante el cambio es más importante que el seguimiento de un plan.

**¿EN QUE CONSISTE XP? SUS OBJETIVOS.**

Los objetivos de XP son muy simples: la satisfacción del cliente. Esta metodología trata de dar al cliente el software que él necesita y cuando lo necesita. Por tanto, debemos responder muy rápido a las necesidades del cliente, incluso cuando los cambios sean al final de ciclo de la programación.

El segundo objetivo es potenciar al máximo el trabajo en grupo. Tanto los jefes de proyecto, los clientes y desarrolladores, son parte del equipo y están involucrados en el desarrollo del software.

**LAS CUATRO VARIABLES**

XP define cuatro variables para proyectos de software: coste, tiempo, calidad y ámbito.

Además de estas cuatro variables, Beck propone que sólo tres puedan ser establecidas por las fuerzas externas (jefes de proyecto y clientes), mientras que el valor de la cuarta variable debe ser establecido por los programadores en función de las otras tres.

XP nos propone que juguemos todas las partes implicadas en el proyecto hasta que el valor que alcancen las cuatro variables sea el correcto para todas las partes.

Además con el agravante de que estas cuatro variables no guardan una relación tan directa como en principio pueda parecer. El incremento del número de programadores no repercutirá de manera lineal en el tiempo de desarrollo del proyecto, siendo de todos conocidos el dicho: *“nueve mujeres no pueden tener un hijo en un mes”.*

Con la calidad suele suceder un fenómeno extraño: frecuentemente un proyecto que tratemos de aumentar la calidad conduce a que el proyecto pueda realizarse en menos tiempo, siempre con unos márgenes obviamente. Es verdad que cuando un equipo de desarrollo se acostumbra a realizar pruebas intensivas, se siguen estándares de codificación, poco a poco se comenzara a andar más rápido y más seguro, por tanto más preparados para futuros cambios, sin estrés y así sucesivamente.

Frente a esto existe la tentación de entregar el trabajo más rápido, por tanto probar menos, codificar más rápido y peor, sin hacer planteamientos maduros, esto repercutirá en la confianza de nuestros clientes, al entregarle trabajos con fallos. Esta es una apuesta a muy corto plazo y suele ser una invitación al desastre, conduce a la desmoralización del equipo, y con ello a la larga a la ralentización del proyecto y la pérdida de tiempo que habríamos conseguido en un principio.

La cuarta variable, el ámbito del proyecto, suele ser conveniente que sea establecida por el equipo de desarrollo. Es una variable muy importante que nos va a decir dónde vamos a llegar con nuestro software, que problemas vamos a resolver y cuales vamos a dejar para siguientes versiones. Cuántas veces hemos escuchado *“Los clientes no nos pueden decir lo que quieren. Cuando le damos lo que nos piden no les gusta”.* Y es que los requisitos nunca son claros al principio y el mismo desarrollo del software hace cambiar los requisitos. Por tanto el ámbito debe de ser dúctil, podremos jugar con él, si el tiempo para el lanzamiento es limitado, siempre habrá cosas que pudramos diferir para siguientes versiones.

Por tanto implementaremos primero los requisitos más importantes para el cliente, de forma que si tenemos que dejar algo para después que sea menos importante que las que ya incorporen un sistema.

**LOS CUATRO VALORES.**

Una de las cosas que a los programadores nos tiene que quedar muy claro es que en el ciclo de vida del desarrollo de un proyecto software los cambios van a aparecer, cambiarán los requisitos, las reglas de negocio, el personal, la tecnología, todo va a cambiar. Por tanto el problema no es el cambio en sí, ya que este va a suceder sino la incapacidad de enfrentarnos a estos cambios.

Como en otra cualquier actividad humana necesitamos valores para desarrollar nuestro trabajo y conseguir los planteamientos iniciales.

Estos cuatro valores son:

1. Comunicación
2. Sencillez
3. Retroalimentación
4. Valentía

**- 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 éstos 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 cómo 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.

**- Sencillez.**

La simplicidad es la base de la programación extrema. Se simplifica el diseño para agilizar el desarrollo y facilitar el mantenimiento.

Siempre debemos hacernos esta pregunta ¿Qué es lo más simple que pueda funcionar? Lograr la sencillez no es fácil. Tenemos cierta tendencia a pensar en qué programaremos mañana, la próxima semana y el próximo mes. Cuantos de nosotros no hacemos a veces más de lo que debemos: “*Ya que estoy tocando esta clase voy a añadirle dos métodos más para visualizar los mensajes en colores*”, cuando eso no está entre los requisitos, “*es que mañana puede que lo necesite*”, si mañana está entre los requisitos, hazlo entonces.

XP nos enseña a apostar, apuesta por hacer una cosa sencilla hoy y pagar un poco más para mañana, si es necesario, que hacer una cosa complicada hoy y no utilizarla después. La sencillez y la comunicación se complementan, cuanto más simple es tu sistema menos tienes que comunicar de él.

**- Retroalimentación.**

*“No me preguntes a mí, pregúntale al sistema”,* es la primera clave de la retroalimentación, por medio de pruebas funcionales a nuestro software este nos mantendrá informado del grado de fiabilidad de nuestro sistema, esta información realmente no tiene precio. Los clientes y las personas que escriben pruebas tienen una retroalimentación real de su sistema.

La retroalimentación actúa junto con la sencillez y la comunicación, cuanto mayor retroalimentación más fácil es la comunicación. Cuanto más simple un sistema más fácil de probar. Escribir pruebas nos orienta como simplificar un sistema, hasta que las pruebas funcionen, cuando las pruebas funcionen tendrá mucho echo.

**- Valentía.**

Asumir retos, ser valientes antes los problemas y afrontarlos.

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 todo lo demás 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 implementaran más fácilmente. Otro ejemplo de valentía es saber cuándo 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.

**LAS CUATRO ACTIVIDADES BASICAS.**

**- Codificar**

Es la única actividad de la que no podremos prescindir. Sin código fuente no hay programa, aunque hay gente que cuenta que existe software en producción del que se perdió el código fuente. Por tanto necesitamos codificar y plasmar nuestras ideas a través del código.

En una programación en XP en pareja el código expresa tu interpretación del problema, así podemos utilizar el código para comunicar, para hacer mías tus ideas, y por tanto para aprender y mejorar.

**- Hacer pruebas**

Las características del software que no pueden ser demostradas mediante pruebas simplemente no existen. Las pruebas me dan la oportunidad de saber si lo que implementé es lo que en realidad yo pensaba que había implementado. Las pruebas nos indican que nuestro trabajo funciona, cuando no podemos pensar en ninguna prueba que pudiese originar un fallo en nuestro sistema entonces has acabado por completo.

No debemos de escribir tan solo una prueba ver que funciona y salir corriendo, debemos de pensar en todas las posibles pruebas para nuestro código, con el tiempo llegaras a conclusiones sobre las pruebas y podrás pensar que si dos de tus pruebas ya funcionan la tercera prueba no será necesaria escribirla, sin caer en demasiada confianza.

Programar y probar es más rápido que sólo programar. Puedes ganar media hora de productividad sin hacer pruebas, pero perderás mucho tiempo en la depuración. Tendrás menos errores, tendrás que volver menos veces sobre el código, te costará menos localizar los errores, perderás menos tiempo escuchado como tus clientes te dicen que no funciona.

Las pruebas deben de ser sensatas y valientes. No podemos hacer pruebecillas que no testen a fondo el sistema, esos agujeros que vamos dejando nos esperan para cuando pasemos de nuevo por allí y volveremos a caer dentro.

**- Escuchar**

Los programadores no lo conocemos todo, y sobre todo muchas cosas que las personas de negocios piensan que son interesantes. Si ellos pudieran programarse su propio software ¿para qué nos querrían?

Si vamos a hacer pruebas tenemos que preguntar si lo obtenido es lo deseado, y tenemos que preguntar a quién necesita la información. Tenemos que escuchar a nuestros clientes cuales son los problemas de su negocio, debemos de tener una escucha activa explicando lo que es fácil y difícil de obtener, y la realimentación entre ambos nos ayudan a todos a entender los problemas.

**- Diseñar**

El diseño crea una estructura que organiza la lógica del sistema, un buen diseño permite que el sistema crezca con cambios en un solo lugar. Los diseños deben de ser sencillos, si alguna parte del sistema es de desarrollo complejo, divídela en varias. Si hay fallos en el diseño o malos diseños, estos deben de ser corregidos cuanto antes.

Tenemos que codificar porque sin código no hay programas, tenemos que hacer pruebas porque sin pruebas no sabemos si hemos acabado de codificar, tenemos que escuchar, porque si no escuchamos no sabemos que codificar ni probar, y tenemos que diseñar para poder codificar, probar y escuchar indefinidamente.

**FASES DE LA METOLOGÍA XP**

**- Planificación.**

**Historias de usuario:** El primer paso de cualquier proyecto que siga la metodología X.P es definir las historias de usuario con el cliente. Las historias de usuario tienen la misma finalidad que los casos de uso pero con algunas diferencias: Constan de 3 ó 4 líneas escritas por el cliente en un lenguaje no técnico sin hacer mucho hincapié en los detalles; no se debe hablar ni de posibles algoritmos para su implementación ni de diseños de base de datos adecuados, etc. Son usadas para estimar tiempos de desarrollo de la parte de la aplicación que describen. También se utilizan en la fase de pruebas, para verificar si el programa cumple con lo que especifica la historia de usuario. Cuando llega la hora de implementar una historia de usuario, el cliente y los desarrolladores se reúnen para concretar y detallar lo que tiene que hacer dicha historia. El tiempo de desarrollo ideal para una historia de usuario es entre 1 y 3 semanas.

**Release planning:** Después de tener ya definidas las historias de usuario es necesario crear un plan de publicaciones, en inglés "Release plan", donde se indiquen las historias de usuario que se crearán para cada versión del programa y las fechas en las que se publicarán estas versiones. Un "Release plan" es una planificación donde los desarrolladores y clientes establecen los tiempos de implementación ideales de las historias de usuario, la prioridad con la que serán implementadas y las historias que serán implementadas en cada versión del programa. Después de un "Release plan" tienen que estar claros estos cuatro factores: los objetivos que se deben cumplir (que son principalmente las historias que se deben desarrollar en cada versión), el tiempo que tardarán en desarrollarse y publicarse las versiones del programa, el número de personas que trabajarán en el desarrollo y cómo se evaluará la calidad del trabajo realizado. (\*Release plan: Planificación de publicaciones).

**Iteraciones:** Todo proyecto que siga la metodología XP. Se ha de dividir en versiones de aproximadamente 3 semanas de duración. Al comienzo de cada versión los clientes deben seleccionar las historias de usuario definidas en el "Release planning" que serán implementadas. También se seleccionan las historias de usuario que no pasaron el test de aceptación que se realizó al terminar la versión anterior. Estas historias de usuario son divididas en tareas de entre 1 y 3 días de duración que se asignarán a los programadores.

**La velocidad del proyecto:** Es una medida que representa la rapidez con la que se desarrolla el proyecto; estimarla es muy sencillo, basta con contar el número de historias de usuario que se pueden implementar en una versión; de esta forma, se sabrá el cupo de historias que se pueden desarrollar en las distintas versiones. Usando la velocidad del proyecto controlaremos que todas las tareas se puedan desarrollar en el tiempo del que dispone la versión. Es conveniente reevaluar esta medida cada 3 ó 4 iteraciones y si se aprecia que no es adecuada hay que negociar con el cliente un nuevo "Release Plan".

**Reuniones diarias**: Es necesario que los desarrolladores se reúnan diariamente y expongan sus problemas, soluciones e ideas de forma conjunta. Las reuniones tienen que ser fluidas y todo el mundo tiene que tener voz y voto.

**- Diseño**

**Metáfora.**

Una metáfora es una historia que todo el mundo puede contar acerca de cómo funciona el sistema. Algunas veces podremos encontrar metáforas sencillas *“Programa de gestión de compras, ventas, con gestión de cartera y almacén”.* Las metáforas ayudan a cualquier persona a entender el objeto del programa.

Desarrollada por los programadores al inicio del proyecto, define una historia de cómo funciona el sistema completo. XP estimula historias, que son breves descripciones de un trabajo de un sistema en lugar de los tradicionales diagramas y modelos UML (Unified Modeling Language). La metáfora expresa la visión evolutiva del proyecto que define el alcance y propósito del sistema. Las tarjetas CRC (Clase, Responsabilidad y Colaboración) también ayudarán al equipo a definir actividades durante el diseño del sistema. Cada tarjeta representa una clase en la programación orientada a objetos y define sus responsabilidades (lo que ha de hacer) y las colaboraciones con las otras clases (cómo se comunica con ellas).

**Diseño sencillo.**

El diseño adecuado para el software es aquel que:

1. Funciona con todas las pruebas.
2. No tiene lógica duplicada.
3. Manifiesta cada intención importante para los programadores.
4. Tiene el menor número de clases y métodos.

Haz el diseño lo más simple posible borra todo lo que puedas sin violar las reglas 1,2 y 3. Contrariamente a lo que se pensaba el *“Implementa para hoy, diseña para mañana”*, no es del todo correcto si piensas que el futuro es incierto.

**- Desarrollo.**

**Recodificación.**

Cuando implementamos nuevas características en nuestros programas nos planteamos la manera de hacerlo lo más simple posible, después de implementar esta característica, nos preguntamos cómo hacer el programa más simple sin perder funcionalidad, este proceso se le denomina recodificar o refactorizar (refactoring). Esto a veces nos puede llevar a hacer más trabajo del necesario, pero a la vez estaremos preparando nuestro sistema para que en un futuro acepte nuevos cambios y pueda albergar nuevas características. No debemos de recodificar ante especulaciones si no solo cuándo el sistema te lo pida.

Refactorizar es mejorar y modificar la estructura y codificación de códigos ya creados sin alterar su funcionalidad. Refactorizar supone revisar de nuevo estos códigos para procurar optimizar su funcionamiento. Es muy común rehusar códigos ya creados que contienen funcionalidades que no serán usadas y diseños obsoletos.

**Programación por parejas.**

Todo el código de producción lo escriben dos personas frente al ordenador, con un sólo ratón y un sólo teclado. Cada miembro de la pareja juega su papel: uno codifica en el ordenador y piensa la mejor manera de hacerlo, el otro piensa más estratégicamente, ¿Va a funcionar?, ¿Puede haber pruebas donde no funcione?, ¿Hay forma de simplificar el sistema global para que el problema desaparezca?

El emparejamiento es dinámico, puedo estar emparejado por la mañana con una persona y por la tarde con otra, si tienes un trabajo sobre un área que no conoces muy bien puedes emparejarte con otra persona que si conozca ese área. Cualquier miembro del equipo se puede emparejar con cualquiera.

**Propiedad colectiva.**

Cualquiera que crea que puede aportar valor al código en cualquier parcela puede hacerlo, ningún miembro del equipo es propietario del código. Si alguien quiere hacer cambios en el código puede hacerlo. Si hacemos el código propietario, y necesitamos de su autor para que lo cambie entonces estaremos alejándonos cada vez más de la comprensión del problema, si necesitamos un cambio sobre una parte del código lo hacemos y punto. XP propone un propiedad colectiva sobre el código nadie conoce cada parte igual de bien pero todos conoce algo sobre cada parte, esto nos preparará para la sustitución no traumática de cada miembro del equipo.

Nadie es el propietario de nada, todos son el propietario de todo. Este método difiere en mucho a los métodos tradicionales en los que un simple programador posee un conjunto de código. Los defensores de XP argumentan que mientras haya más gente trabajando en una pieza, menos errores aparecerán.

**Integración continúa.**

El código se debe integrar como mínimo una vez al día, y realizar las pruebas sobre la totalidad del sistema. Una pareja de programadores se encargara de integrar todo el código en una máquina y realizar todas las pruebas hasta que estas funcionen al 100%.

Permite al equipo hacer un rápido progreso implementando las nuevas características del software. En lugar de versiones estables de acuerdo a un cronograma establecido, los equipos de programadores XP pueden reunir su código y reconstruir el sistema varias veces al día. Esto reduce los problemas de integración comunes en proyectos largos y estilo cascada.

**40 Horas semanales.**

Si queremos estar frescos y motivados cada mañana y cansado y satisfecho cada noche. El viernes quiero estar cansado y satisfecho para sentir que tengo dos días para pensar en algo distinto y volver el lunes lleno de pasión e ideas. Esto requiere que trabajemos 40 horas a la semana, mucha gente no puede estar más de 35 horas concentrada a la semana, otros pueden llegar hasta 45 pero ninguno puede llegar a 60 horas durante varias semanas y aun seguir fresco, creativo y confiado. Las horas extras son síntoma de serios problemas en el proyecto, la regla de XP dice nunca 2 semanas seguidas realizando horas extras.

La programación extrema sostiene que los programadores cansados escriben código de menor cualidad. Minimizar las horas extras y mantener los programadores frescos, generará código de mayor calidad. Como dice Beck, está bien trabajar tiempos extra cuando es necesario, pero no se ha de hacer durante dos semanas seguidas.

**Cliente In-situ.**

Un cliente real debe sentarse con el equipo de programadores, estar disponible para responder a sus preguntas, resolver discusiones y fijar las prioridades. Lo difícil es que el cliente nos ceda una persona que conozca el negocio para que se integre en el equipo normalmente estos elementos son muy valiosos, pero debemos de hacerles ver que será mejor para su negocio tener un software pronto en funcionamiento, y esto no implica que el cliente no pueda realizar cualquier otro trabajo.

**Estándares de codificación.**

Si los programadores van a estar tocando partes distintas del sistema, intercambiando compañeros, haciendo refactoring, debemos de establecer un estándar de codificación aceptado e implantado por todo el equipo.

Define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona.

**- Pruebas**

**Hacer pruebas.**

No debe existir ninguna característica en el programa que no haya sido probada, los programadores escriben pruebas para chequear el correcto funcionamiento del programa, los clientes realizan pruebas funcionales. El resultado un programa más seguro que conforme pasa el tiempo es capaz de aceptar nuevos cambios.

Uno de los pilares de la metodología XP es el uso de test para comprobar el funcionamiento de los códigos que vayamos implementando. El uso de los test en X.P es el siguiente:

1. Se deben crear las aplicaciones que realizarán los test con un entorno de desarrollo específico para test.
2. Hay que someter a tests las distintas clases del sistema omitiendo los métodos más triviales.
3. Se deben crear los test que pasarán los códigos antes de implementarlos; en el apartado anterior se explicó la importancia de crear antes los test que el código.
4. Un punto importante es crear test que no tengan ninguna dependencia del código que en un futuro evaluará.
5. Como se comentó anteriormente los distintos test se deben subir al repositorio de código acompañados del código que verifican.
6. Test de aceptación. Los test mencionados anteriormente sirven para evaluar las distintas tareas en las que ha sido dividida una historia de usuario.

Al ser las distintas funcionalidades de nuestra aplicación no demasiado extensas, no se harán test que analicen partes de las mismas, sino que las pruebas se realizarán para las funcionalidades generales que debe cumplir el programa especificado en la descripción de requisitos.

**Conclusión.**

Es la metodología de desarrollo de la ingeniería de software más destacada de los procesos ágiles de desarrollo de software. La programación extrema se diferencia de las metodologías tradicionales principalmente en que hace más énfasis en la adaptabilidad que en la previsibilidad.

Los objetivos de XP son muy simples: la satisfacción del cliente y el segundo objetivo es potenciar al máximo el trabajo en grupo.

Posee 4 variables: coste, tiempo, calidad y ámbito, 4 valores: Comunicación, sencillez, retroalimentación y valentía, y 4 actividades: codificar, hacer pruebas, escuchar y diseñar.

Y por último las fases de la metodología XP se basan en la planificación, diseño, desarrollo y hacer pruebas.

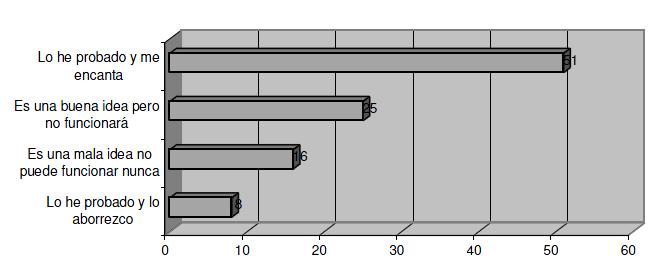
**Bibliografía.**

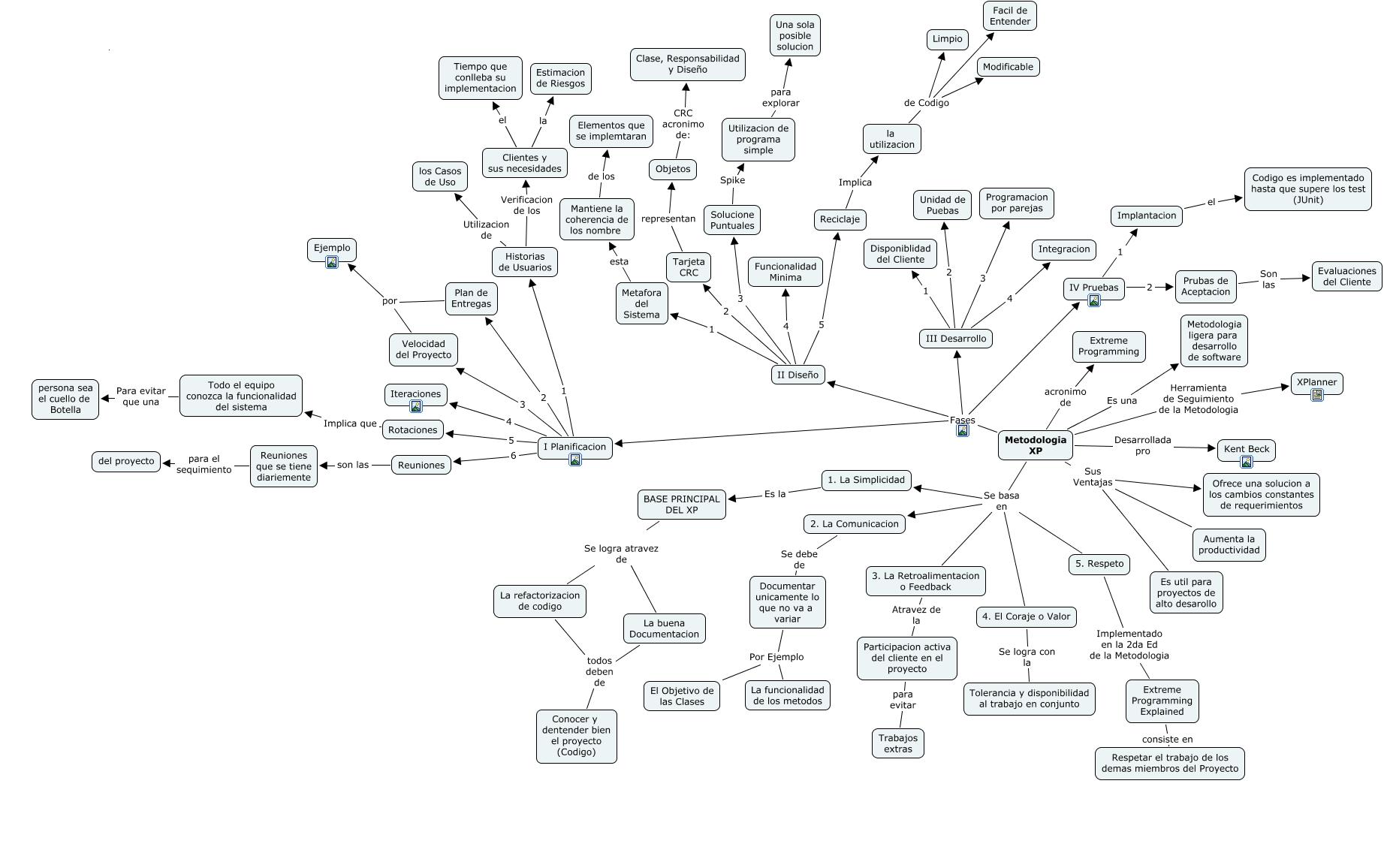
http://wiki.monagas.udo.edu.ve/index.php/Metodolog%C3%ADas\_SCRUM\_y\_XP

http://procesosdesoftware.wikispaces.com/METODOLOGIA+XP

http://www.willydev.net/descargas/prev/ExplicaXP.pdf

**Anexos.**





http://2.bp.blogspot.com/_oZAEdvGyPyM/TS2_pgimDaI/AAAAAAAAACc/yuvE29Ab9lk/s1600/NI%25C3%2591OS+ESPECIALES.JPG

UNIVERSIDAD DE ORIENTE

NUCLEO ANZOATEGUI

ESCUELA DE INGENIERIA Y CIENCIAS APLICADAS

DEPARTAMENTO DE COMPUTACION Y SISTEMAS

**Profesor: Bachiller:**

Víctor Mujica Gabriel Farro

C.I.: 20.762.197

Barcelona, 22/01/2013.