UNIVERSIDAD NACIONAL EXPERIMENTAL DE LOS LLANOS OCCIDENTALES EZEQUIEL ZAMORA
Ingeniería en Informática
Subproyecto: Metodología de Desarrollo del Software
Semestre VII

METODOLOGÍA ACTUAL

METODOLOGÍA XP

Bachilleres:
Bustamante Dayana C.I: 22.983.709
Rodríguez Jean C. C.I: 21.169.047

Barinas, Marzo del 2014

INTRODUCCIÓN

Las metodologías de desarrollo de software son marcos o modelos de trabajos que se utilizan para construir, planificar y controlar el proceso de desarrollo de sistemas.

Hoy en día existen infinidades de metodologías para desarrollar software. Entre ellas encontramos las Metodologías Tradicionales, las Metodologías Iterativas/Evolutivas, las Metodologías basadas en Tecnología Web, y las Metodologías Ágiles.

Mediante el siguiente trabajo nos enfocaremos en las Metodologías Ágiles, y se profundizara en la XP (Programación Extrema), la cual pertenece a este grupo de metodologías. A continuación, estudiaremos su origen, características, valores, ventajas, desventajas, pasos y fases de desarrollo de la Metodología XP según su creador Kent Beck (1999).

Además, se estudiara la normativa de calidad del software, y nos enfocaremos en el modelo ISO 9126, el cual se enfoca en la calidad del producto que se desarrollara.

METODOLOGÌAS ÁGILES

Existen numerosas propuestas de metodología para desarrollar software. Tradicionalmente estas metodologías se centran en el control del proceso, estableciendo rigurosamente las actividades, herramientas y notaciones al respecto, dado estas reglas estas metodologías se caracterizan por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas. Este enfoque no resulta ser el más adecuado para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. En este escenario, las metodologías ágiles emergen como una posible respuesta para llenar ese vacío metodológico.

Los objetivos de las metodologías ágiles, entre los cuales se destaca la preferencia de algunos valores por sobre otros, por ejemplo: Individuos e interacciones, sobre procesos y herramientas, Software operativo, sobre documentación extensiva Y Colaboración con el cliente, sobre negociación de Contratos.

METODOLOGÍA XP
Según Kent Beck 1999

ORIGEN DE LA METODOLOGÍA XP

La programación extrema o eXtreme Programming (XP) es una metodología de desarrollo de la ingeniería de software formulada 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. Se puede considerar la programación extrema como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo con el proyecto, y aplicarlo de manera dinámica durante el ciclo de vida del software.

Kent Beck, Creador de la Metodología XP

CARACTERÍSTICAS DE LA METODOLOGÍA XP

- Se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad.
- Se aplica de manera dinámica durante el ciclo de vida del software.
- Es capaz de adaptarse a los cambios de requisitos.
- Los individuos e interacciones son más importantes que los procesos y herramientas.
- Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las 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.

- Software que funcione es más importante que documentación exhaustiva.
- 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.

- La colaboración con el cliente es más importante que la negociación de contratos.

- 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.

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

VALORES DE LA METODOLOGÍA XP

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 auto-documentado. 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 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.

- **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 programador esa 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 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.

PASOS DE LA METODOLOGÍA XP

Los Pasos fundamentales inmersos en las fases del método son:

- **Desarrollo iterativo e incremental:** Pequeñas mejoras, unas tras otras.

- **Pruebas unitarias continuas:** Son frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.

- **Programación en parejas:** Se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. Se supone que 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 compartido:** 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 del 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.

FASES DE LA METODOLOGÍA XP

Según Kent Beck 1999

Fase I - Planificación del proyecto

- **Historias de usuario:**

El primer paso de cualquier proyecto que siga la metodología XP 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 X.P. se ha de dividir en iteraciones de aproximadamente 3 semanas de duración. Al comienzo de cada iteración 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 iteración 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 iteración; de esta forma, se sabrá el cupo de historias que se pueden desarrollar en las distintas iteraciones. Usando la velocidad del proyecto controlaremos que todas las tareas se puedan desarrollar en el tiempo del que dispone la iteración. 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".

- **Programación en Parejas:**

La metodología X.P. aconseja la programación en parejas pues incrementa la productividad y la calidad del software desarrollado.

El trabajo en pareja involucra a dos programadores trabajando en el mismo equipo; mientras uno codifica haciendo hincapié en la calidad de la función o método que está implementando, el otro analiza si ese método o función es adecuado y está bien diseñado. De esta forma se consigue un código y diseño con gran calidad.

- **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.

Fase II - Diseño

- **Diseños Simples:**

La metodología XP sugiere que hay que conseguir diseños simples y sencillos. Hay que procurar hacerlo todo lo menos complicado posible para conseguir un diseño fácilmente entendible e impleméntable que a la larga costará menos tiempo y esfuerzo desarrollar.

- **Glosarios de Términos:**

Usar glosarios de términos y una correcta especificación de los nombres de métodos y clases ayudará a comprender el diseño y facilitará sus posteriores ampliaciones y la reutilización del código.

- **Riesgos:**

Si surgen problemas potenciales durante el diseño, XP sugiere utilizar una pareja de desarrolladores para que investiguen y reduzcan al máximo el riesgo que supone ese problema.

- **Funcionabilidad extra**:

Nunca se debe añadir funcionalidad extra al programa aunque se piense que en un futuro será utilizada. Sólo el 10% de la misma es utilizada, lo que implica que el desarrollo de funcionalidad extra es un desperdicio de tiempo y recursos.

- **Refactorizar:**

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.

Fase III - Codificación

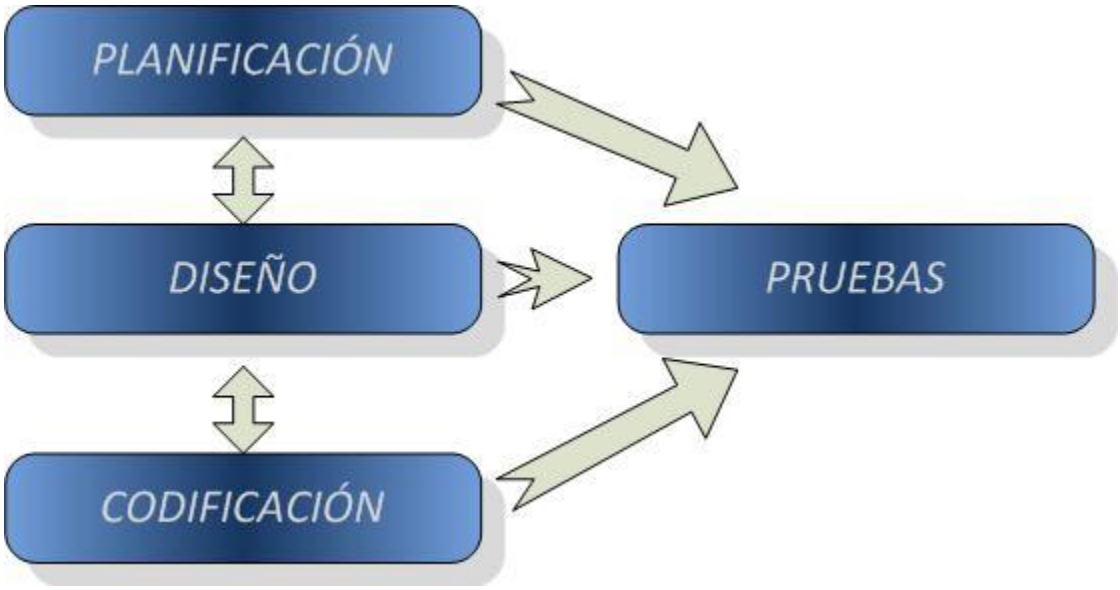
Como ya se dijo en la introducción, el cliente es una parte más del equipo de desarrollo; su presencia es indispensable en las distintas fases de XP. A la hora de codificar una historia de usuario su presencia es aún más necesaria. No olvidemos que los clientes son los que crean las historias de usuario y negocian los tiempos en los que serán implementadas. Antes del desarrollo de cada historia de usuario el cliente debe especificar detalladamente lo que ésta hará y también tendrá que estar presente cuando se realicen los test que verifiquen que la historia implementada cumple la funcionalidad especificada. La codificación debe hacerse ateniendo a estándares de codificación ya creados. Programar bajo estándares mantiene el código consistente y facilita su comprensión y escalabilidad.

Fase IV - Pruebas

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 XP es el siguiente:

- Se deben crear las aplicaciones que realizarán los test con un entorno de desarrollo específico para test.
- Hay que someter a tests las distintas clases del sistema omitiendo los métodos más triviales.

- 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.
- Un punto importante es crear test que no tengan ninguna dependencia del código que en un futuro evaluará.
- Como se comentó anteriormente los distintos test se deben subir al repositorio de código acompañados del código que verifican.
- 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.



Fases de la Metodología XP

VENTAJAS Y DESVENTAJAS DE LA METODOLOGÍA XP

- Ventajas:

- Programación organizada.
- Menor taza de errores.
- Satisfacción del programador.

- Desventajas:

- Es recomendable emplearlo solo en proyectos a corto plazo.
- Altas comisiones en caso de fallar.

NORMATIVA DE CALIDAD TIC EN EL SOFTWARE

Existen diversos modelos de calidad en el ámbito del software. Podríamos agruparlos en sistemas de gestión, calidad en el producto software y calidad en los procesos software. Normalmente las normativas de calidad son conjuntos de buenas prácticas que se aplican sobre el ciclo de vida de proyectos informáticos y que contribuyen a mejorar los factores de la calidad del software que se han expuesto con anterioridad.

Existen multitud de modelos para la gestión de la calidad del software y otros sistemas y normas de gestión que se han aplicado sobre estos procesos, muchas de ellas con apéndices (normas específicas) para uno de los conceptos más importante en el software: la evaluación.

Además el mundo del software englobado en los servicios TI puede ser evaluado en calidad según otros sistemas de gestión sobre TI. En este sentido algunas empresas de desarrollo de software han implantado sistemas de gestión basados en ISO 9001, ISO 27001 o ISO 20000 con alcances en los procesos de desarrollo y entrega, pero éstos quizás no son la mejor opción en el caso de que el corazón productivo de la organización se únicamente el software.

Modelo ISO 9126

La ISO 9126 es un estándar internacional para la evolución de Software. El estándar está dividido en cuatro partes las cuales dirigen, respectivamente, lo siguiente: modelo de calidad, métricas externas, métricas internas y calidad en las métricas de uso.

Este estándar está pensado para los desarrolladores, adquirentes, personal que asegure la calidad y evaluadores independientes, responsables de especificar y evaluar la calidad del producto software.

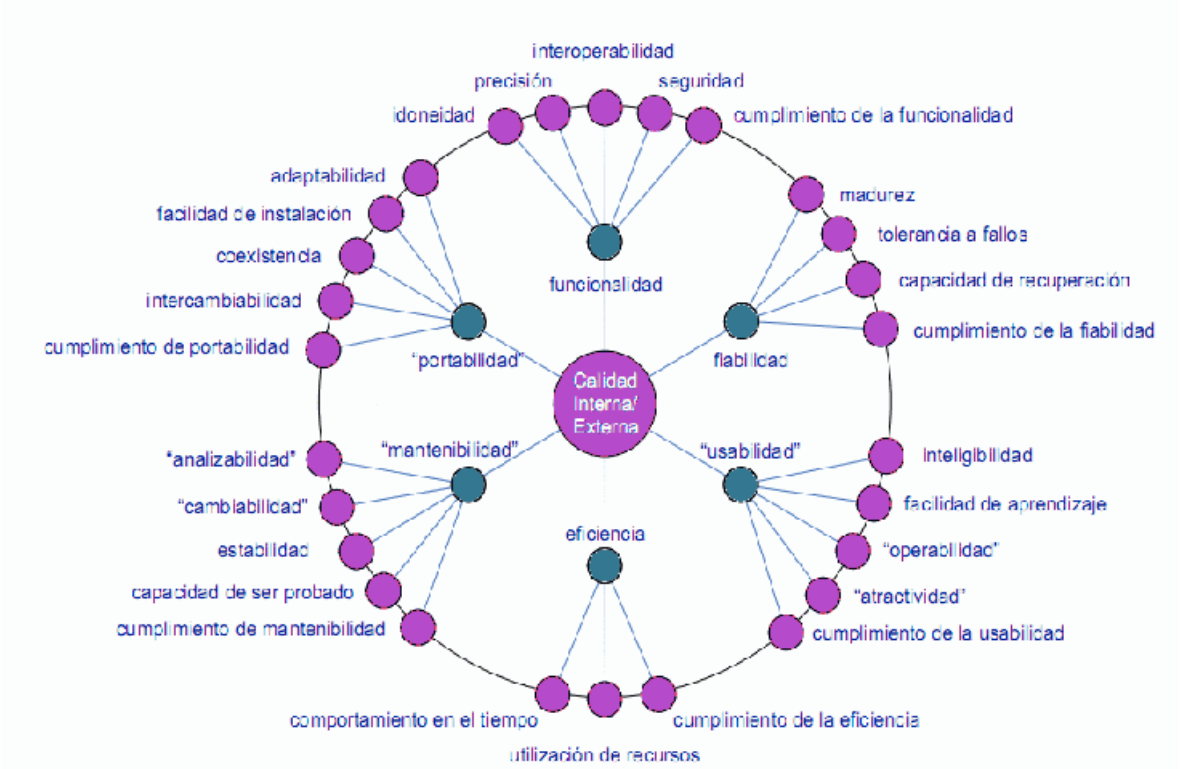
Por tanto, puede servir para validar la completitud de una definición de requisitos, identificar requisitos de calidad de software, objetivos de diseño y prueba, criterios de aseguramiento de la calidad, etc.

Este estándar proviene desde el modelo establecido en 1977 por McCall y sus colegas, los cuales propusieron un modelo para especificar la calidad del software.

ISO 9126 distingue entre fallos y no conformidad, siendo un fallo el no cumplimiento de los requisitos previos, mientras que la no conformidad afecta a los requisitos especificados. Una distinción similar es hecha entre la validación y la verificación.

Características del Modelo ISO 9126

El modelo establece diez características, seis que son comunes a las vistas interna y externa y cuatro que son propias de la vista en uso. Las características que definen las vistas interna y externa, se muestran a continuación en la figura y son:



• Funcionalidad, capacidad del software de proveer los servicios necesarios para cumplir con los requisitos funcionales.

• Fiabilidad, capacidad del software de mantener las prestaciones requeridas del sistema, durante un tiempo establecido y bajo un conjunto de condiciones definidas.

• Usabilidad, esfuerzo requerido por el usuario para utilizar el producto satisfactoriamente.

• Eficiencia, relación entre las prestaciones del software y los requisitos necesarios para su utilización.

• Mantenibilidad, esfuerzo necesario para adaptarse a las nuevas especificaciones y requisitos del software.

• Portabilidad, capacidad del software ser transferido de un entorno a otro

Mientras que las características propias de la vista en uso, se muestran a continuación en la siguiente figura:

• Efectividad, capacidad del software de facilitar al usuario alcanzar objetivos con precisión y completitud.

• Productividad, capacidad del software de permitir a los usuarios gastar la cantidad apropiada de recursos en relación a la efectividad obtenida.

• Seguridad, capacidad del software para cumplir con los niveles de riesgo permitidos tanto para posibles daños físicos como para posibles riesgos de datos.

• Satisfacción, capacidad del software de cumplir con las expectativas de los usuarios en un contexto determinado.

CONCLUSIÓN

Actualmente existen muchas propuestas de metodología para desarrollar software, sin embargo el uso exclusivo de alguna metodología dependerá, del que sea de preferencia para el desarrollador de acuerdo al tipo de sistema que este desarrollara.

Las metodologías ágiles, tradicionalmente se centran en el control del proceso, en el establecimiento de las actividades, herramientas y notaciones al respecto, caracterizándose de esta forma como una metodología rígida, dirigida por la documentación de cada actividad implementada.

Dentro del marco de metodologías agiles que existen, encontramos a la muy reconocida y usada Metodología XP (Programación Extrema), propuesta por Kent Beck en 1999. Esta metodología constituye un modelo de trabajo compartido, además existe la conexión entre cliente y desarrollador, lo que permitirá la construcción del sistema de acuerdo a los requerimientos establecidos por el cliente en un principio.

Esperamos que haya aprendido valiosa información, de nuestro trabajo de investigación.

Existen numerosas metodologías

Metodologías Ágiles

Se caracteriza por ser rígida

Se centran en el control del proceso

Características de XP

Se aplica de manera dinámica

Es capaz de adaptarse a los cambios de requisitos

Los individuos e interacciones son más importantes

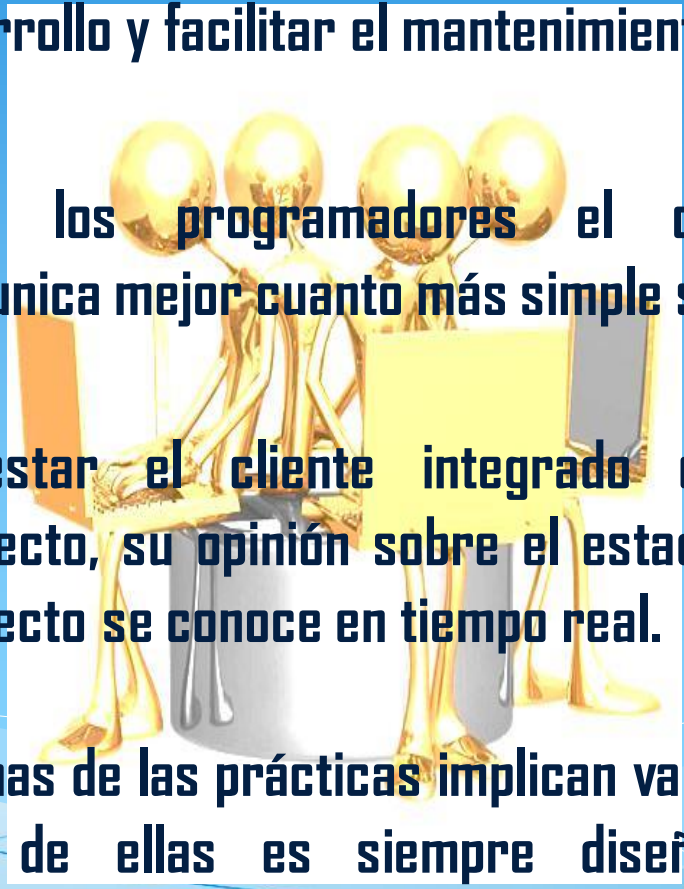
Es más importante que el Software funcione

La colaboración con el cliente es más importante

Valores de XP

Simplicidad → Se simplifica el diseño para agilizar el desarrollo y facilitar el mantenimiento.

Comunicación → Para los programadores el código comunica mejor cuanto más simple sea.

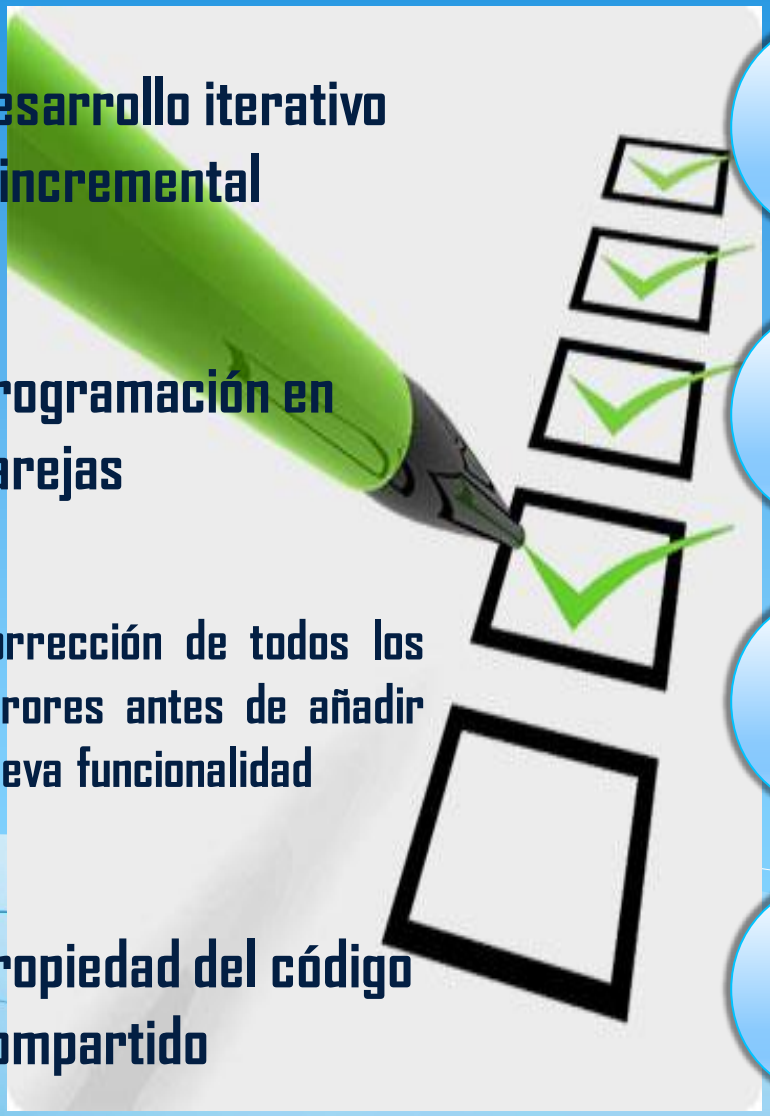


Retroalimentación → Al estar el cliente integrado en el proyecto, su opinión sobre el estado del proyecto se conoce en tiempo real.

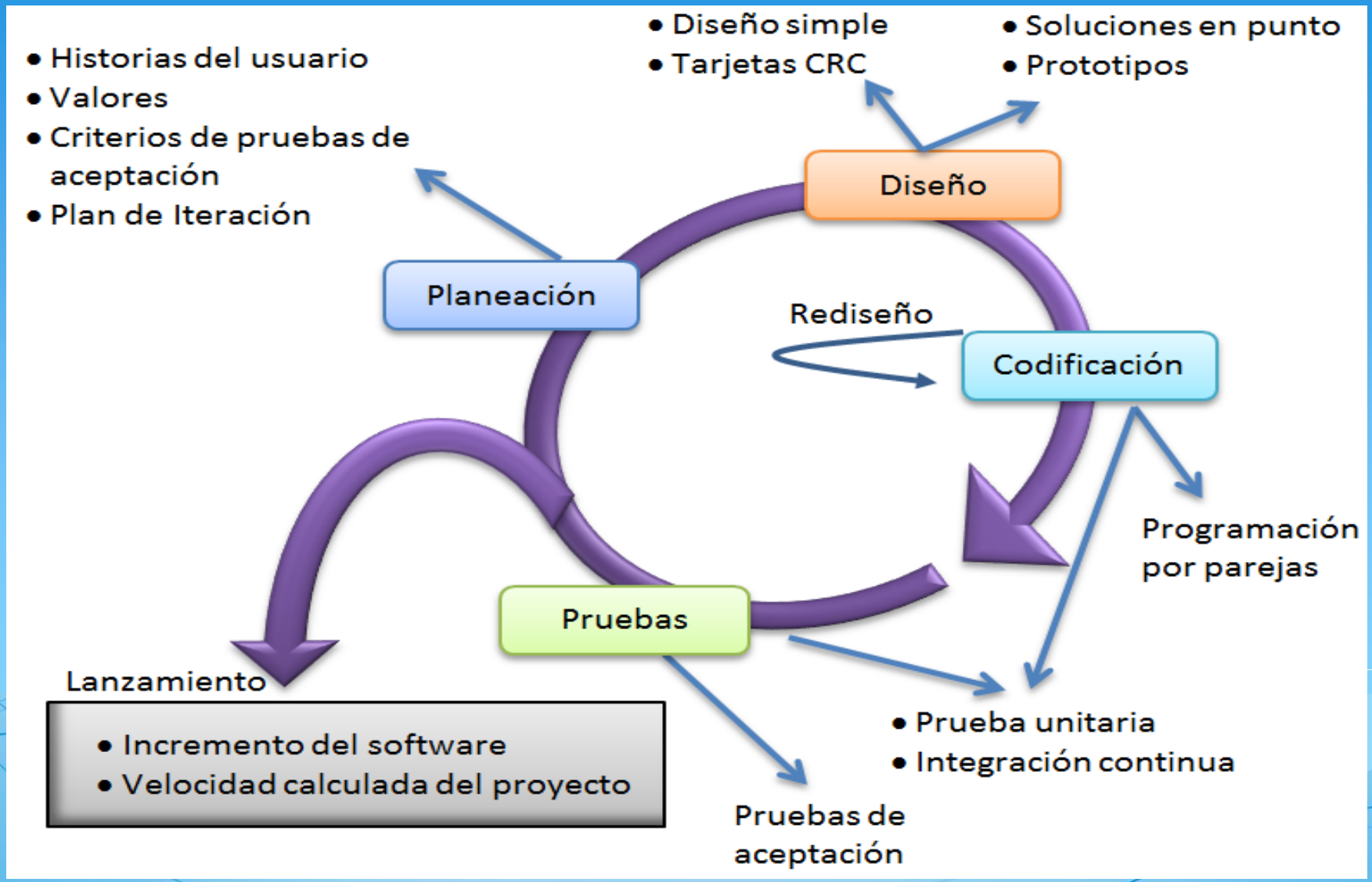
Coraje o Valentía → Muchas de las prácticas implican valentía. Una de ellas es siempre diseñar y programar para hoy y no para mañana.

Pasos de la Metodología XP

1. Desarrollo iterativo e incremental
2. Pruebas unitarias continuas
3. Programación en parejas
4. Frecuente integración del equipo de programación con el cliente
5. Corrección de todos los errores antes de añadir nueva funcionalidad
6. Refactorización del código
7. Propiedad del código compartido
8. Simplicidad del código



Fases de la Metodología XP



Fase I – Planificación del Proyecto

1. Historias de usuario
2. Release Planning
3. Iteraciones
4. La Velocidad del Proyecto
5. Programación en Parejas
6. Reuniones Diarias

Fase II – Diseño

1. Diseños Simples
2. Glosarios de Términos
3. Riesgos
4. Funcionabilidad extra
5. Refactorizar

Fase III – Codificación

A la hora de codificar una historia de usuario su presencia es aún más necesaria.

Fase IV – Pruebas

Uno de los pilares de la metodología XP es el uso de test para comprobar el funcionamiento de los códigos que vayamos implementando.

1. Se deben crear las aplicaciones que realizarán los test
2. Someter a tests las distintas clases del sistema
3. Crear los test que pasarán los códigos antes de implementarlos
4. Crear test que no tengan ninguna dependencia del código
5. Se deben subir al repositorio de código
6. Test de aceptación
7. Las pruebas se realizarán para las funcionalidades generales

Ventajas de XP

- Programación organizada
- Menor taza de errores
- Satisfacción del programador

Desventajas de XP

- Es recomendable emplearlo solo en proyectos a corto plazo.
- Altas comisiones en caso de fallar.

Normativa de Calidad TIC

Norma ISO 9126